Langage C

Table des matières

Le C est un langage de programmation impératif, généraliste, conçu pour la programmation système. Inventé au début des années 1970 pour réécrire UNIX, C est devenu un des langages les plus utilisés. De nombreux langages plus modernes comme C++, Java et PHP reprennent des aspects de C.

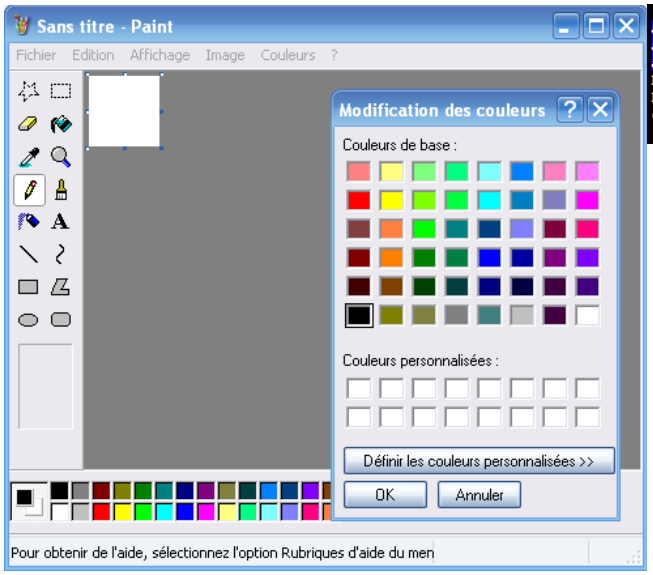
1. Premiers pas

1.1. Console ou fenêtre ?

Il existe deux types de programmes :

- les programmes avec fenêtres ;
- les programmes en console.

Fenêtre



Console

Les programmes console ont été les premiers à apparaître. À cette époque, l'ordinateur ne gérait que le noir et blanc et il n'était pas assez puissant pour créer des fenêtres comme on le fait aujourd'hui.

1.2. Mon premier programme

Nous allons écrire un programme en mode console qui affichera « Hello world! ».

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

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

Les premières lignes qui commencent par # sont des directives de préprocesseur qui seront lues par un programme appelé préprocesseur, un programme qui se lance au début de la compilation.

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

Ces lignes demandent d'inclure deux fichiers au projet, c'est-à-dire d'ajouter deux fichiers pour la compilation : stdio.h et stdlib.h.

Ces fichiers sont des fichiers source tout prêts qu'on les appelle bibliothèques qui contiennent la

plupart des fonctions de base dont a besoin un programme. stdio.h en particulier contient la fonction printf.

int main()

main est la fonction principale du programme, c'est toujours par la fonction main que le programme commence. C'est la seule qui soit obligatoire, aucun programme ne peut être compilé sans elle.

Une fonction a un début et une fin, délimités par des accolades { et } et retourne ou non une valeur. Toute la fonction main se trouve donc entre ces accolades.

Les lignes à l'intérieur d'une fonction s'appellent des instructions.

Toute instruction se termine obligatoirement par un point-virgule « ; ».

printf("Hello world!\n"); demande à afficher le message « Hello world! » à l'écran.

return 0; la fonction main retourne la valeur 0 de type entière (int).

En pratique, 0 signifie « tout s'est bien passé » et n'importe quelle autre valeur signifie « erreur ».

Pour utiliser une fonction, il suffit d'écrire son nom, suivi d'une parenthèse ouvrante, puis éventuellement de paramètres séparés par une virgule, suivi d'une parenthèse fermante et d'un point-virgule.

ma_fonction(parametre_1, paramètre2) ;

Le caractère \n fait partie des caractères spéciaux qui commence par un anti-slash (\ ), et le second un nombre ou une lettre. Exemple :

- \n : retour à la ligne
- \t : tabulation

1.3. Les commentaires

Un commentaire permet d'ajouter des annotations dans le codes source et de mieux vous expliquer à quoi peut servir telle ou telle ligne de code.

Si après avoir lu uniquement les commentaires d'un programme vous n'en comprenez pas le fonctionnement, jetez le !

Il y a plusieurs manières d'insérer un commentaire selon la longueur du commentaire.

// Ceci est un commentaire sur une ligne

ou

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

Exemple :

```
/*
Ci-dessous, ce sont des directives de préprocesseur.
Ces lignes permettent d'ajouter des fichiers au projet,
fichiers que l'on appelle bibliothèques contiennent des fonctions toutes prêtes.
*/
#include <stdio.h>
#include <stdlib.h>

/*
Ci-dessous, vous avez la fonction principale du programme, appelée main.
C'est par cette fonction que tous les programmes commencent.
*/
int main()
{
      printf("Bonjour");      // Cette instruction affiche Bonjour à l'écran
      return 0;          // Le programme renvoie le nombre 0 puis s'arrête
}
```

Lors de la compilation, tous les commentaires seront ignorés. Ces commentaires n'apparaîtront pas dans le programme final, ils servent seulement aux programmeurs.

2. Les variables

2.1. Déclarer une variable

En langage C, une variable est constituée :

- d'une valeur : par exemple le nombre qu'elle stocke.
- d'un nom : c'est ce qui permet de la manipuler.

Un nom de variable ne peut contenir que des lettres et des chiffres et doit commencer par une lettre ; les espaces et accents sont interdits et le langage est sensible à « la casse »[1].

Pour manipuler une variable, il faut la déclarer et indiquer son type. Voici les principaux types de variables existant en langage C :

Nom du type	**Minimum**	**Maximum**
signed char	-127	128
int	-32 767	32 768
long	-2 147 483 647	2 147 483 640
float	-1.10^{37}	1.10^{37}
double	-1.10^{37}	1.10^{37}

Les trois premiers types (signed char, int, long) permettent de stocker des nombres entiers : 1, 2, 3, 4… alors que les deux derniers (float, double) permettent de stocker des nombres décimaux (appelés nombres flottants) : 13.8, 16.911…

Pour les types entiers (signed char, int, long…), il existe d'autres types dits unsigned (non signés) qui eux ne peuvent stocker que des nombres positifs. Pour les utiliser, il suffit d'écrire le mot

1 Distinction majuscules/minuscules

unsigned devant le type :

type	valeurs	Taille (octet)
unsigned char	0 à 255	1
unsigned int	0 à 65 535	2
unsigned long	0 à 4 294 967 295	4

Pour déclarer plusieurs variables du même type, il vous suffit de séparer les différents noms de variables par des virgules sur la même ligne.

Attention : une variable n'est jamais initialisée avec une valeur par défaut, elle prend la valeur qui se trouvait là avant dans la mémoire, et cette valeur peut être n'importe quoi !

2.2. Les constantes

Les constantes sont des variables particulières dont la valeur reste constante.

Pour déclarer une constante, il faut utiliser le mot const juste devant le type de la variable et lui donner obligatoirement une valeur au moment de sa déclaration.

Ex : const float PI = 3.14159265359;

Par convention on écrit les noms des constantes entièrement en majuscules afin de distinguer facilement les constantes des variables.

2.3. Les chaînes de caractères

Le type char permet de stocker des nombres allant de -128 à 127, unsigned char des nombres de 0 à 255. On peut donc utiliser le type char pour stocker UNE lettre (codée sur 1 octet dans la table ASCII[2]).

Une chaîne de caractères représente alors un tableau de type char que l'on notera :

- char chaine[6]; // tableau qui contient 6 caractères
- char chaine[] = "salut"; // tableau dimensionné pour contenir le texte "salut"
- char *chaine = "salut"; // pointeur sur le texte "salut"

Une chaîne de caractère doit impérativement contenir un caractère spécial à la fin de la chaîne, appelé « caractère de fin de chaîne ». Ce caractère s'écrit '\0' pour que votre ordinateur sache quand s'arrête la chaîne.

```
chaine[0] = 's';
chaine[1] = 'a';
chaine[2] = 'l';
chaine[3] = 'u';
chaine[4] = 't';
chaine[5] = '\0';
```

Par conséquent, pour stocker le mot « salut » (qui comprend 5 lettres) en mémoire, il ne faut pas un tableau de 5 char, mais de 6 !

En tapant entre guillemets la chaîne que vous voulez mettre dans votre tableau, le compilateur C calcule automatiquement la taille nécessaire. C'est-à-dire qu'il compte les lettres et ajoute 1 pour placer le caractère \0.

Remarque : pour initialiser un type char, on peut directement affecter la variable avec la lettre (chaine[1] = 'a') ou avec son code ASCII (chaine[1] = 97[3]).

2 American Standard Code for Information Interchange (Code américain normalisé pour l'échange d'information)
3 en valeur décimale (chaine[1] = 0x61 en hexadécimal)

2.4. Afficher le contenu d'une variable

La fonction printf() peut afficher à l'écran le contenu d'une variable en accord avec le format passé en paramètre.

Le format de conversion est indiqué par une chaîne de caractères composée par le caractère % suivi d'une lettre qui décrit le type de variable attendu.

Format	**Type attendu**	
%c	char	caractère
%s	char *	chaîne de caractères
%d	int	entier
%ld	long	entier long
%f	float	flottant
%lf	double	flottant long

```
int main()
{
	int	date = 1949, nb_dec = 2037;
	char	*nom = "John von Neumann" ;

	// En 1949, John von Neumann a obtenu 2037 décimales de π
	printf("En %d, %s a obtenu %d décimales de π\n", date, nom, nb_dec);

	return 0;
}
```

2.5. Récupérer une saisie

La fonction scanf() analyse une saisie au clavier conformément au format de ce que l'utilisateur doit entrer (int, float, ...). Lorsque le programme arrive à un scanf, il se met en pause et attend que l'utilisateur entre une valeur qui sera stockée dans la variable.

```
int main()
{
	char	nom[21] = "";		// On initialise la variable à une chaine vide
	int	age = 0;		// On initialise la variable à 0

	printf("Comment vous appellez-vous ? ");
	// les caractères saisis pointent directement sur le tableau
	scanf("%s", nom);

	printf("Quel age avez-vous ? ");
	// le caractère & fait pointer la saisie sur la variable age
	scanf("%d", &age);

	printf("Bonjour %s, vous avez %d ans\n", nom, age);

	return 0;
}
```

Ce type de saisie n'est cependant pas exsangue de tout problème :

- si vous rentrez un nombre décimal, comme 2.9, il sera automatiquement tronqué (seule la partie entière sera conservée).
- si vous tapez des lettres pour l'âge, la variable ne changera pas de valeur.
- Si vous tapez trop de lettres (> 20) pour le nom, le programme plantera...

3. Les opérations de base

Opération	**Signe**	**Remarque**
Addition	+	
Soustraction	-	
Multiplication	*	
Carré	**	
Division	/	
Modulo	%	Reste de la division entière
Décalage gauche[4]	<<	Multiplication par 2
Décalage droite	>>	Division par 2
AND	&	Opération bit à bit
OR	|	
XOR	^	
NOT	!	
Complément à 1	~	

Attention à respecter les types lors des opérations : diviser deux entiers donnera un entier même si la variable qui reçoit le résultat est un flottant.

```
int main()
{
	int	a = 5, b= 2;
	float	c = (float) a / (float) b;		// conversion des int en float (technique du cast)

	printf("%.2f\n", c);	// on affiche avec 2 chiffres après la virgule

	return 0;
}
```

3.1. Raccourcis d'écriture

Il existe en C des techniques permettant de raccourcir l'écriture des opérations.

Au lieu d'écrire : nombre = nombre + 1;

on écrira : nombre++;

4 Voir cours 'Systèmes numériques'

De même pour : nombre = nombre - 1;

on pourra écrire : nombre--;

Il existe d'autres raccourcis qui fonctionnent sur le même principe qui fonctionnent pour toutes les opérations de base :

```
int nombre = 2;
nombre += 4;          // nombre vaut 6...
nombre -= 3;          // ... nombre vaut maintenant 3
nombre *= 5;          // ... nombre vaut 15
nombre /= 3;          // ... nombre vaut 5
nombre %= 3;          // ... nombre vaut 2 (car 5 = 1 * 3 + 2)
```

3.2. La bibliothèque mathématique

En langage C, il existe ce qu'on appelle des bibliothèques « standard », c'est-à-dire des bibliothèques toujours utilisables. Ce sont en quelque sorte des bibliothèques « de base » qu'on utilise très souvent.

La bibliothèque math contient de nombreuses fonctions mathématiques toutes prêtes.

Pour pouvoir utiliser les fonctions de la bibliothèque mathématique, il est indispensable de mettre la directive de préprocesseur suivante en haut du programme : **#include** <math.h>

Voici quelques fonctions principales :

fonction	**description**
fabs[5]	Cette fonction retourne la valeur absolue d'un nombre flottant, c'est-à-dire |x|.
ceil	Cette fonction renvoie le premier nombre entier après le nombre décimal qu'on lui donne.
floor	C'est l'inverse de la fonction précédente : elle renvoie le nombre directement en dessous.
pow	Cette fonction permet de calculer la puissance d'un nombre.
sqrt	Cette fonction calcule la racine carrée d'un nombre.
sin, cos, tan	Ce sont les trois fonctions sinus, cosinus et tangente utilisées en trigonométrie. Ces fonctions attendent une valeur en radians.
asin, acos, atan	Ce sont les fonctions arc sinus, arc cosinus et arc tangente utilisées en trigonométrie.
exp	Cette fonction calcule l'exponentielle d'un nombre.
log	Cette fonction calcule le logarithme népérien d'un nombre.
log10	Cette fonction calcule le logarithme base 10 d'un nombre.

5 Équivalent à la fonction abs de la librairie standard stdlib.h (ne foctionne que sur des entiers)

4. Les conditions

Une condition peut être écrite en langage C sous différentes formes. On parle de structures conditionnelles.

4.1. if… else

Les symboles suivant permettent de faire des comparaisons simples ou multiples pour les conditions.

Symbole	Signification
==	Est égal à
>	Est supérieur à
<	Est inférieur à
>=	Est supérieur ou égal à
<=	Est inférieur ou égal à
!=	Est différent de

Symbole	Signification
AND	ET logique
&&	ET logique
OR	OU logique
||	OU logique
!	NON logique

Pour introduire une condition :

1. on utilise le mot if, qui en anglais signifie « si ».
2. on ajoute à la suite entre parenthèses la condition en elle-même.
3. on ouvre des accolades à l'intérieur desquelles on placera les instructions à exécuter si la condition est **vraie**.
4. **Facultativement** : on utilise le mot else, qui en anglais signifie « sinon ».
5. puis on ouvre des accolades à l'intérieur desquelles on placera les instructions à exécuter si la condition est **fausse**.

```
int main()
{
	int	age = 8;				// initialisation de la variable

	if ( age < 12 )			// test
	{
		printf("Salut gamin !");		// action si condition vraie
	}
	else	// SINON
	{
		// action si condition fausse
	}

	return 0;
}
```

Dans le cas des conditions multiples, on utilise les opérateurs logiques.

```
int main()
```

```
{
	int		age = 8;		// initialisation de la variable
	unsigned char	garcon = 0;		// équivalent d'un booléen initialisé à faux

	if ( age < 12 && garcon )		// si j'ai moins de 12 ans et que je suis un garçon
	{
		printf("Bonjour Jeune homme");
	}
	else
	 {
		if ( age < 12 && !garcon )	// si j'ai moins de 12 ans et que je suis pas un garçon
		{
			printf("Bonjour Mademoiselle");
		}
	}

	return 0;
}
```

4.2. switch... case

Les structures à base de if… else suffisent pour traiter n'importe quelle condition comme le montre l'exemple suivant :

```
int main()
{
	char *jour = "mercredi";

	// début de la semaine
	if ( !strcmp(jour, "lundi") || !strcmp(jour, "mardi") )
	    printf("courage !!!");
	// milieu de la semaine
	else if ( !strcmp(jour, "mercredi") )
	    printf("c'est le jour des enfants");
	// fin de la semaine
	else if ( !strcmp(jour, "jeudi") || !strcmp(jour, "vendredi") )
	    printf("bientôt le we !");
	else    // il faut traiter les autres jours (cas par défaut)
	    printf("vive le week end !");

    return 0;
}
```

Cependant, pour une imbrication de if... else trop importante, il existe une autre structure plus souple : switch... case.

```
int main()
{
	const int	LUNDI = 0, MARDI = 1, MERCREDI = 2, JEUDI = 3, VENDREDI = 4,
			SAMEDI = 5, DIMANCHE = 6;
	int		jour = MERCREDI;
	char		*texte;

	switch ( jour )	// on indique sur quelle variable on travaille
```

```
	{
		case 0 :	// dans le cas où c'est le début de la semaine
		case 1 :	// on peut tester deux valeurs à la suite
			texte = "courage !!!";
		break; // ne pas oublier « break » sinon le langage C continue

		case 2 :	 // dans le cas où c'est le milieu de la semaine
			texte = "c'est le jour des enfants";
		break;

		case 3 :	// dans le cas où c'est la fin de la semaine
		case 4 :
			texte = "bientôt le we !";
		break;

		default:	// il faut traiter les autres jours (cas par défaut)
			texte = "vive le week end !";
		break;
	}

	printf("%s\n", texte);

	return 0;
}
```

Le mot-clé default est le traitement par défaut quelle que soit la valeur de la variable.

Le langage C traite les instructions des case à la suite. Pour interrompre le traitement, il faut utiliser break.

4.3. Les ternaires : des conditions condensées

Un ternaire est une condition condensée qui fait deux choses sur une seule ligne :

- on teste la valeur d'une variable dans une condition ;
- on affecte une valeur à une variable selon que la condition est vraie ou non.

C'est l'écriture condensée de if… else.

```
int main()
{
	int			age = 24;
	unsigned char		majeur;

	if ( age >= 18 )
	{
		majeur = 1;
	}
	else
	{
		majeur = 0;
	}

	return 0;
}
```

ou encore plus simplement :

```c
int main()
{
	int 		age = 24;
	unsigned char	majeur =  ( age >= 18 ) ? 1 : 0 ;

	return 0;
}
```

5. Les boucles

5.1. Tant que : while

Une boucle permet de répéter des instructions plusieurs fois.

- les instructions sont d'abord exécutées dans l'ordre, de haut en bas
- à la fin des instructions, on retourne à la première
- et ainsi de suite...

Tant que la condition est remplie, les instructions sont réexécutées. Dès que la condition n'est plus remplie, on sort de la boucle.

```c
int main()
{
	int	nombre_de_lignes = 1;

	// on boucle tant que on n'est pas arrivé à 100 lignes
	while ( nombre_de_lignes <= 100 )
	{
		printf("ligne n°%d\n", nombre_de_lignes); // affiche le n° de ligne
		nombre_de_lignes++;			// incrément du nombre de ligne
	}

	return 0;
}
```

Il faut TOUJOURS s'assurer que la condition sera fausse au moins une fois. Si elle ne l'est jamais, alors la boucle s'exécutera à l'infini !

5.2. Faire tant que : do... while

Les boucles do-while ressemblent beaucoup aux boucles while, mais l'expression est testée à la fin de chaque itération plutôt qu'au début. La principale différence par rapport à la boucle while est que la première itération de la boucle do-while est toujours exécutée au moins une fois (l'expression n'est testée qu'à la fin de l'itération).

```c
int main()
{
	int	nombre_de_lignes = 1;

	do	// on boucle tant que on n'est pas arrivé à 100 lignes
	{
		printf("ligne n°%d\n", nombre_de_lignes); // affiche le n° de ligne
		nombre_de_lignes++;			// incrément du nombre de ligne
	} while ( nombre_de_lignes <= 100 );
```

```
        return 0;
}
```

5.3. Boucle : for

La boucle for est un autre type de boucle, dans une forme un peu plus condensée et plus commode à écrire, ce qui fait que for est assez fréquemment utilisé.

for (initialisation ; condition ; incrémentation)

1. initialisation. C'est la valeur que l'on donne au départ à la variable.
2. condition. Comme pour le while, tant que la condition est remplie, la boucle est réexécutée. Dès que la condition ne l'est plus, on en sort.
3. incrémentation, qui vous met à jour la variable à chaque tour de boucle.

```
int main()
{
        int     nombre_de_lignes;

        for (nombre_de_lignes = 1 ; nombre_de_lignes <= 100 ; nombre_de_lignes++)
        {
                printf("ligne n°%d\n", nombre_de_lignes);    // affiche le n° de ligne
        }

        return 0;
}
```

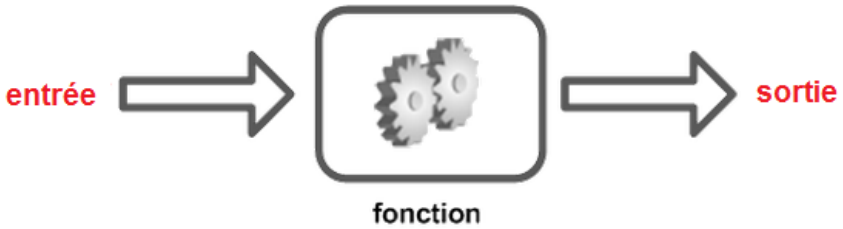
Cette boucle est utilisée lorsque l'on connaît à l'avance le nombre d'instructions à répéter.

6. Les fonctions

Une fonction est une série d'instructions qui effectue des actions qui sont répétées plusieurs fois dans le code sans avoir à les réécrire à chaque fois.

Lorsqu'on appelle une fonction, il y a trois étapes.

1. L'entrée: on donne des informations à la fonction en lui passant des paramètres avec lesquelles travailler.
2. Le traitement : grâce aux informations qu'elle a reçues en entrée.
3. La sortie : une fois qu'elle a fini son traitement, la fonction renvoie un résultat.



La syntaxe pour coder un fonction est donc :

```
type nomFonction(parametres)
{
```

```
        // Insérez vos instructions ici
}
```

- Le type (correspond à la sortie) : résultat renvoyé par la fonction.

  → **char, int, double**, … si la fonction renvoie un résutat

  → **void** si la fonction ne renvoie rien

- nomFonction : c'est le nom de la fonction (pas d'accents, pas d'espaces, etc.)
- parametres (correspond à l'entrée) : valeurs avec lesquelles la fonction va travailler. Il peut ne pas y avoir de paramètres.

Remarque : Les parenthèses sont obligatoires, quand bien même votre fonction n'attendrait aucun paramètre.

Exemple : on veut afficher le cube des nombres de 1 à 10.

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

int cube(int valeur)     // chaque paramètre doit être typé
{ //-- début de la fonction
      // fonction :     calcule le cube d'une valeur passée en paramètre
      // in :           valeur, valeur à calculer
      // out:           résultat du paramètre élevé au cube

      return valeur * valeur * valeur;    // il faut utiliser le mot-clef return
} //-- fin de la fonction

//---- programme principal
int main()
{
      int   i;

      for (i = 1; i <= 10; i++)
      {
            printf("la valeur de %d^3 est %d\n", i, cube(i));     // appel de la
fonction
      }

      return 0;
}
```

Une fonction peut recevoir plusieurs paramètres séparés par des virgules.

```
int puissance(int valeur, int exposant)
{
      // fonction :     calcule la valeur d'un nombre élevé à une puissance
      // in :           valeur, valeur à calculer ; exposant, puissance
      // out:           la valeur élevée à la puissance

      int   i, total = 1;
      for (i = 1; i <= exposant; i++)
            total *= valeur;

      return total;
}
```

6.1. Fonctions de manipulation des chaînes

Les chaînes de caractères sont fréquemment utilisées. Le langage C fournit dans la bibliothèque string.h une pléthore de fonctions dédiées aux calculs sur des chaînes.

Il existe de nombreuses fonctions toutes prêtes de manipulation des chaînes dans la bibliothèque string. Voici les principales :

fonction	**description**
strlen	size_t strlen(const char* chaine); Cette fonction retourne la longueur de la chaîne (sans compter le caractère \0).
strcpy	char* strcpy(char* copieDeLaChaine, const char* chaineACopier); La fonction strcpy permet de copier une chaîne à l'intérieur d'une autre. La fonction renvoie un pointeur sur copieDeLaChaine
strcat	char* strcat(char* chaine1, const char* chaine2); Cette fonction ajoute une chaîne à la suite d'une autre. La fonction retourne un pointeur vers chaine1.
strcmp	int strcmp(const char* chaine1, const char* chaine2); strcmp compare 2 chaînes entre elles. strcmp renvoie : • 0 si les chaînes sont identiques ; • une autre valeur (positive ou négative) si les chaînes sont différentes.
strchr	char* strchr(const char* chaine, int caractereARechercher); La fonction strchr recherche un caractère dans une chaîne. La fonction renvoie un pointeur vers le premier caractère qu'elle a trouvé, c'est-à-dire qu'elle renvoie l'adresse de ce caractère dans la mémoire. Elle renvoie NULL si elle n'a rien trouvé.
strrchr	identique à strchr, sauf que celle-là renvoie un pointeur vers le dernier caractère qu'elle a trouvé dans la chaîne plutôt que vers le premier.
strpbrk	char* strpbrk(const char* chaine, const char* lettresARechercher); Recherche un des caractères dans la liste donnée sous forme de chaîne, contrairement à strchr qui ne peut rechercher qu'un seul caractère à la fois. La fonction renvoie un pointeur vers le premier de ces caractères qu'elle y a trouvé.
strstr	char* strstr(const char* chaine, const char* chaineARechercher); Cette fonction recherche la première occurrence d'une chaîne dans une autre chaîne. Elle renvoie un pointeur quand elle a trouvé ce qu'elle cherchait. Elle renvoie NULL si elle n'a rien trouvé.
sprintf	écrire dans une chaîne. Cette fonction ressemble énormément au printf mais, au lieu d'écrire à l'écran, sprintf écrit dans… une chaîne

6.2. Les fonctions C

Le langage C propose des centaines et des centaines de fonctions prédéfinies. La documentation du langage C les répertorie toutes, classées par catégories.

Il faut toujours vérifier qu'une fonction n'existe pas avant de l'écrire.

7. Les pointeurs

Une donnée est toujours stockée en mémoire. La valeur stockée dans une variable possède donc une adresse[6] et une seule. On ne peut pas stocker deux nombres par adresse.

Le langage C permet d'accéder à la valeur d'une variable ou à son adresse.

Pour afficher l'adresse de la variable, on doit utiliser le format %ld ou %p (ce dernier affichera la valeur au format hexadécimal) dans le printf. En outre, on doit envoyer à la fonction printf non pas la variable, mais son adresse… Pour cela, il faut mettre le symbole & devant la variable.

```
int main()
{
	int	i = 10;

	printf("adresse %p : valeur %d\n", &i, i);

	return 0;
}
```

7.1. Utilisation de pointeurs

Pour créer une variable de type pointeur, il faut rajouter le symbole * devant le nom de la variable.

Pour initialiser un pointeur, c'est-à-dire lui donner une valeur par défaut, on n'utilise généralement pas le nombre 0 mais le mot-clé NULL (en majuscules) : ce pointeur ne contient aucune adresse.

```
// Créer une variable de type int dont la valeur vaut 10
int	age = 10;

// Créer une variable de type pointeur sur un int dont la valeur vaut l'adresse de la variable age
int	*ptrAge = &age;
```

Remarque : le pointeur doit pointer sur une variable du type correspondant à sa déclaration.

On accède à la valeur pointée par un pointeur en le préfixant du symbole *.

```
int main()
{
	int	age = 10;
	int	*ptrAge = &age;

	// les valeurs doivent être identiques !
	printf("adresse de la variable age %p et du pointeur %p\n", &age, ptrAge);

	// et là aussi !
	printf("valeur de la variable age %d et du pointeur %d\n", age, *ptrAge);

	return 0;
}
```

Pour le pointeur ptrAge :

- ptrAge signifie : « Je veux la valeur de ptrAge » (cette valeur étant une adresse),
- * ptrAge signifie : « Je veux la valeur de la variable qui se trouve à l'adresse contenue dans

6 L'adresse est un nombre qui représente la position dans la mémoire de l'ordinateur.

ptrAge ».

7.2. Passer un pointeur à une fonction

Un des gros intérêts des pointeurs est qu'on peut les envoyer à des fonctions pour qu'ils modifient directement une variable en mémoire, et non une copie. En effet, une fonction ne peut renvoyer qu'une valeur. Pour récupérer les modifications sur plusieurs variables, il est impératif d'utiliser des pointeurs.

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

/* Attention à ne pas oublier de mettre une étoile devant le nom
    des pointeurs. Il faut modifier la valeur des variables,
    et non leur adresse ! */
void decoupeMinutes(int* ptrHeures, int* ptrMinutes)
{
      // fonction :     transforme des minutes au format heures:minutes
      // in :           ptrHeures, les heures ;  ptrMinutes, les minutes
      // out:           rien

      *ptrHeures  = *ptrHeures / 60;
      *ptrMinutes = *ptrMinutes % 60;
}

//---- programme principal
int main()
{
      int   heures = 0, minutes = 90;

      // On envoie l'adresse de heures et minutes
      decoupeMinutes(&heures, &minutes);

      // Les deux valeurs ont été modifiées !
      printf("%d heures et %d minutes\n", heures, minutes);

      return 0;
}
```

En résumé :

1. Les variables heures et minutes sont créées dans le main.
2. On envoie à la fonction decoupeMinutes l'adresse de heures et minutes.
3. La fonction decoupeMinutes récupère ces adresses dans des pointeurs appelés ptrHeures et ptrMinutes.
4. La fonction decoupeMinutes modifie directement les valeurs des variables heures et minutes en mémoire car elle possède leurs adresses dans des pointeurs.

8. Les tableaux

Les tableaux sont une suite de variables de même type, situées dans un espace contigu en mémoire. Un tableau a une dimension bien précise.

8.1. Déclarer un tableau

Pour déclarer un tableau, il faut respecter la syntaxe suivante :

<type du tableau> <nom du tableau> [] = { <contenu du tableau>};

Ex : int tableau[4]; // tableau de 4 entiers

Un tableau commence à l'indice n° 0 !

On peut initialiser un tableau de deux façon :

Élément par élément	À la déclaration

int tableau[4];
tableau[0] = 10;
tableau[1] = 23;
tableau[2] = 505;
tableau[3] = 8;
	int tableau[4] = {10, 23, 505, 8};

Pour initialiser tout le tableau à 0, il suffit d'initialiser au moins la première valeur à 0, et toutes les autres valeurs non indiquées prendront la valeur 0.

int tableau[4] = {0}; // Tous les éléments du tableau seront initialisées à 0

De même, int tableau[4] = {10, 23}; // Valeurs insérées : 10, 23, 0, 0

Les deux premiers éléments du tableau seront initialisés et tous les autres prendront la valeur 0 (par défaut).

8.2. Parcourir un tableau

Pour accéder à un élément du tableau, il suffit de donner son indice dans le tableau.

```
int main()
{
	int	tableau[4] = {10, 23, 505, 8};
	int	i ;

	for (i = 0 ; i < 4 ; i++)
	{
		printf("élément n° %d : %d\n", i, tableau[i]);
	}

    return 0;
}
```

Ou d'utiliser le pointeur du tableau :

```
int main()
{
	int	tableau[4] = {10, 23, 505, 8};
	int	i;

	for (i = 0 ; i < 4 ; i++)
	{
		// on affiche la valeur pointée, puis on incrémente le pointeur
		printf("élément n° %d : %d\n", i, *tableau++);
	}
```

```
    return 0;
}
```

Ainsi l'instruction tableau[2] est équivalente à *(tableau + 2).

Remarque : l'instruction printf("%d\n", tableau); donne l'adresse en mémoire du 1er élément du tableau.

8.3. Passage de tableaux à une fonction

Il va falloir envoyer deux informations à la fonction : l'adresse du tableau et sa taille.

```
void affiche(int *tableau, int tailleTableau)
{
	int	i;

	for (i = 0 ; i < tailleTableau ; i++)
	{
		printf("valeur élément n°%d : %d\n", i, tableau[i]);
	}
}
```

Il existe une autre façon d'indiquer que la fonction reçoit un tableau plutôt qu'un pointeur sur un entier en la déclarant de la façon suivante :

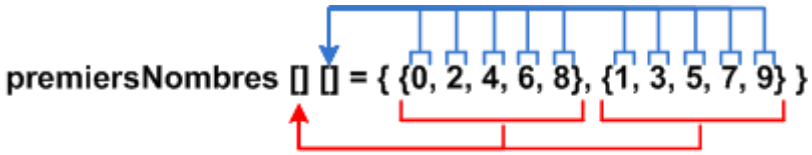
void affiche(int tableau[], int tailleTableau)

Cela revient exactement au même, mais la présence des crochets permet de voir que c'est un tableau que la fonction prend, et non un simple pointeur.

8.4. Tableaux multi dimensionnels

Un tableau multidimensionnel n'est rien d'autre qu'un tableau contenant au minimum deux tableaux.

Chaque dimension est désignée par des crochets [].



Nous changeons de colonne par le biais de la première paire de crochets
Nous choisissons le terme d'un tableau grâce à la deuxième paire de crochets

```
int main()
{
	int	premiersNombres[2][5] =		// tableau 2 dimension : 2 x 5
		{
			{0, 2, 4, 6, 8},  // ne pas oublier la virgule !
			{1, 3, 5, 7, 9}
		};

	int	i, j;

	for (i = 0 ; i < 2 ; i++)
	{
		for (j = 0 ; j < 5 ; j++)
		{
			printf("élément n° (%d,%d) : %d\n", i, j, tableau[i][j]);
```

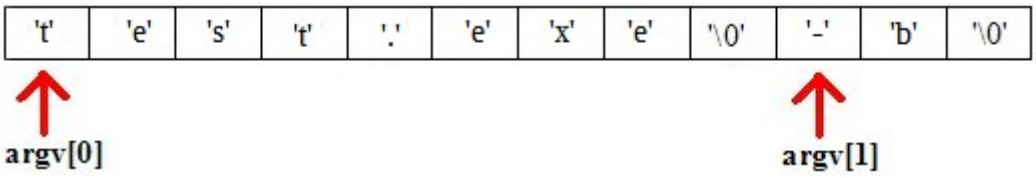
```
            }
        }

        return 0;
}
```

9. Les paramètres de la fonction main

Un programme en langage C commence souvent par :

```
int main (int argc, char *argv[])
```

Le paramètre argv est un tableau de pointeurs. Chacun de ces pointeurs pointe sur des chaînes de caractères :



Le paramètre argc indique le nombre de chaînes de caractères sur lequel pointe argv.

Ces chaînes de caractères sont en réalité envoyées par le système d'exploitation (Windows, GNU/Linux, etc...) au programme au lancement de ce dernier.

La première cellule de argv pointe sur une chaîne de caractères qui indique la commande utilisée pour lancer le programme (ex : test.exe). Par ailleurs, la norme veut que argc soit strictement supérieur à 0 et que argv[argc] pointe sur NULL.

Les autres cellules peuvent pointer sur d'autres chaînes de caractères envoyées par le système d'exploitation qui sont généralement indiquées par l'utilisateur en ligne de commande, grâce à la console.

```
test.exe -b 9600
```

Chaque mot après le nom du programme sera placé dans une chaîne de caractères à laquelle vous pourrez accéder grâce à argv.

Ainsi, dans l'exemple ci-dessus, on aura :

- dans argv[0] : test.exe
- dans argv[1] : -b
- dans argv[2] : 9600

Ainsi, on peut facilement récupérer ces arguments dans le programme C :

```
int main(int argc, char *argv[])
{
    int i;

    for (i=0; i < argc; i++)
        printf("Argument %d : %s\n", i+1, argv[i]);

    return EXIT_SUCCESS;
}
```

TEST

1 - Comment s'appelle le programme chargé de traduire un code source en binaire ?

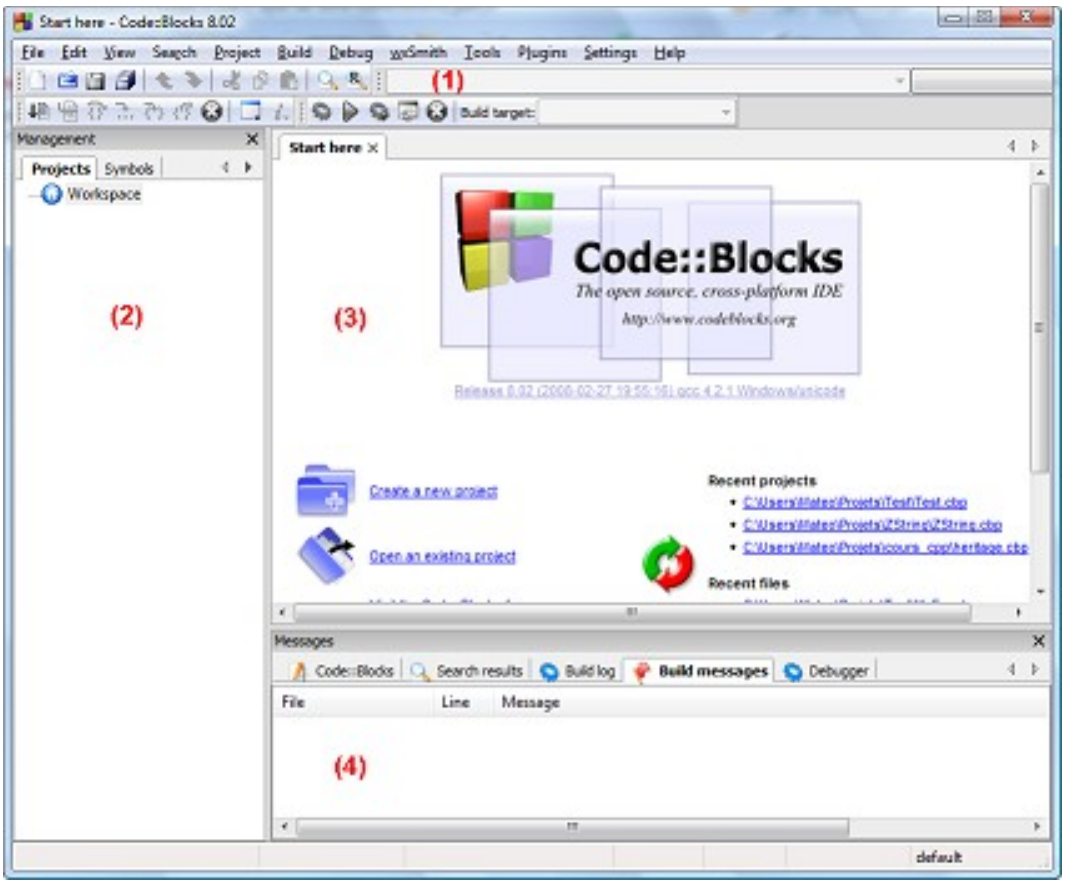
- Le diffuseur
- Le compilateur
- Le binarisateur

2 - Lequel de ces programmes n'est pas un IDE ?

- Bloc-Notes
- Microsoft Visual Studio
- Code::Blocks
- Xcode

3 - Où s'afficheront les erreurs de compilation dans Code::Blocks d'après la capture suivante ?

- Zone (1)
- Zone (2)
- Zone (3)
- Zone (4)



4- Que signifie le return 0 dans le code suivant ?

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

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

}

- Le programme a effectué 0 action
- Le programme ne s'est pas bien exécuté
- Le programme s'est bien exécuté

5 - Quel symbole permet d'effectuer un retour à la ligne à l'écran ?

- \t
- \n
- Il suffit d'appuyer sur la touche Entrée, voyons !

6 - Si ma variable "compteEnBanque" est un long qui vaut 6 500 000 (soyons fous), qu'est-ce que cette ligne de code affichera à l'écran ?

printf("Vous avez %ld euros sur votre compte", compteEnBanque);

- Vous avez %ld euros sur votre compte
- Vous avez 6 500 000 euros sur votre compte
- Vous avez ld euros sur votre compte, compteEnBanque

7 - Combien vaudra la variable "resultat" après cette opération ?

resultat = (8 / 3) - 2;

- -2
- 0
- 1
- 2

8- Quel est le problème de ce switch ?

```
switch ( variable )
{
	case 5:
		printf("Salut");
	case 12:
		printf("Bonjour");
	default:
		printf("Au revoir");
}
```

- Il manque des instructions break
- Il faut mettre un point-virgule à la fin du switch
- Il faut ouvrir des accolades pour chaque "case"
- C'est "case default" et pas "default" tout court

9 - Laquelle de ces boucles for pourrait afficher les messages suivants dans la console ?

Ligne n°1

Ligne n°3
Ligne n°5
Ligne n°7

- for (compteur = 1 ; compteur < 9 ; compteur += 2)
- for (compteur = 1 ; compteur <= 7 ; compteur++)
- for (compteur = 0 ; compteur < 9 ; compteur += 2)
- for (compteur = 1 ; compteur < 8 ; compteur++)

10 - Combien de fois le message "Salut" sera-t-il affiché ici ?

```
int compteur = 14;
while (compteur < 15)
{
        printf("Salut\n");
}
```

- 0 fois
- 1 fois
- 14 fois
- 15 fois
- C'est une boucle infinie

11 - Dans quel cas l'instruction return n'est pas obligatoire ?

- Quand la fonction ne prend aucun paramètre en entrée
- Quand la fonction est de type void
- Quand la fonction doit renvoyer 0

12 - Que sont les paramètres d'une fonction ?

- Des indications sur le nom de la fonction
- Des indications sur la valeur qu'elle doit renvoyer
- Des variables qu'on lui envoie pour qu'elle puisse travailler

13 - Quel est le problème de cette fonction qui est censée calculer le carré de la variable nombre ?

Rappel : le carré d'un nombre, c'est nombre * nombre (le nombre multiplié par lui-même)

```
int carre(int nombre){
        int resultat = 0;
        resultat = nombre * nombre;
}
```

- La fonction ne retourne aucune valeur
- La fonction ne marche pas car on a oublié un point-virgule quelque part
- Quel problème ? Il n'y a pas de problème !