Langage C

Table des matières

1. Introduction

C est un langage de programmation impératif inventé au début des années 1970 pour réécrire UNIX.

Ci-dessous un exemple de programme qui permet d'afficher du texte :

```
#include <stdio.h>
int main(){
  puts("Hello, World!");
  return 0;
}
```

Exemple 1: Programme basique en C

On remarque que :

- '#include' permet d'inclure des fichiers ;
- la fonction 'main' est le point d'entrée du programme;

2. Les différents types de fichiers

a) Code source

Les fichiers qui contiennent le code source, c'est à dire les fonctions ou méthodes et l'initialisation des variables, possèdent l'extension « .c ».

b) Entêtes

Les fichiers d'entêtes ne contiennent pas de code source mais uniquement les prototypes des fonctions et / ou l'instanciation des variables. Ces fichiers possèdent l'extension « .h ».

3. Syntaxe

a) Les commentaires

Il est possible d'insérer des commentaires dans votre code C des façons suivantes :

```
// Ceci est un commentaire;
/*
* Il est possible de faire des
* commentaires de plusieurs
* lignes
*/
```

Exemple 2: Commentaires en C

b) Affichage de texte

Il est possible d'insérer du texte grâce aux fonctions « echo » et « print » de la manière suivante :

```
printf("Bonjour %s", "Magali") ;
puts("Ceci est une chaîne de caractères") ;
```

Exemple 3: Insertion de texte

4. Inclusion

L'inclusion permet d'insérer le contenu d'un fichier, généralement un fichier d'en-tête, dans un autre fichier. L'inclusion se fait par le biais de l'instruction pré-processeur « #include ».

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

Exemple 4: Utilisation de « #include »

5. Les variables

a) Types

La caractéristique la plus importante d'un langage de programmation est certainement la faculté de manipuler des variables. C permet de manipuler les styles suivants :

- les nombres : int, double, float ;
- les caractères : char ;
- les booléens : bool ;

Il existe également un type de données trivial : NULL.

b) Spécificateurs

Les spécificateurs de type permettent de changer / affiner la propriété d'un type :

- short : 2 octets
- long : 4 octets
- signed : le MSB désigne le signe (nombre positif et négatif) ;
- unsigned : le MSB fait partie du nombre (nombre positif uniquement) ;

c) Mémorisation

La mémorisation permet de spécifier dans quelle zone mémoire se trouve la variable :

- auto : positionnement sur la pile (stack) ;
- static : positionnement dans le tas (heap) ;
- register : positionnement dans un registre processeur ;
- extern : déclaration dans un fichier externe ;

d) Qualificateur

Le qualificateur permet de spécifier le caractère modifiable d'une variable :

- const : la variable n'est pas modifiable ;
- volatile : modification indépendante des instructions du programme ;

e) Déclarateur

Le déclarateur correspond au nom de la variable. Il n'est pas possible d'attribuer n'importe quel nom à une variable et il faut respecter les règles suivantes :

- les variables ne peuvent contenir les caractères : +, -, %, (,), &, ... ;
- toujours faire commencer le nom par une lettre ou « _ » ;

Exemple de déclaration ou initialisation :

```
const static shot int i ;
```

Exemple 5: Déclaration de variables

Le fait de stocker une valeur dans une variable s'appelle **l'affectation**.

```
const static shot int i;
i = 2;
```

Exemple 6: Affectation de variables

L'initialisation et l'affectation peuvent se faire en même temps.

Lorsque l'on parle d'affectation ou distingue deux parties :

- la « lvalue » qui se situe à gauche (l → *left*) ;
- la « rvalue » qui se situe à droite (r → *right*) ;

L'affectation consiste à positionner la valeur de la « lvalue » avec le résultat de la « rvalue ».

6. Les pointeurs et références

Les pointeurs sont des variables qui contiennent l'adresse d'autres variables. Ils sont très utiles pour manipuler des variables complexes (tableaux ou structures), ou lorsque l'on cherche à modifier la valeur d'une variable située dans la « heap ».

Les références sont un moyen, à partir d'une variable initialisée, d'obtenir l'adresse de cette variable. Une référence s'obtient grâce au symbole « & ».

Ci-dessous un exemple ou l'on instancie une variable ainsi qu'un pointeur vers cette variable.

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

int main(void) {
	int i = 1;
	int * p_int = &i;

	printf("La valeur de i est: %i\n", i); //affiche 1
	printf("L'adresse de i est: %p\n", &i); //affiche l'adresse de i eg. 0x7fff19efa01c
	printf("La valeur pointée par p_int est: %i\n", *p_int); //affiche 1
	printf("La valeur de p_int est: %p\n", p_int); //affiche l'adresse de i eg. 0x7fff19efa01c
	printf("L'adresse de p_int est: %p\n", &p_int); //affiche l'adresse de p_int eg. 0x7fff19efa010
	return EXIT_SUCCESS;
}
```

Exemple 7: Création d'une variable et d'un pointeur

Attention, un pointeur n'est pas obligé de pointer vers une adresse mémoire « valide » ! Tant que l'opération de référencement n'est pas faite, le pointeur ne pointe vers aucune valeur particulière (valeur poubelle). Il faut procéder au référencement du pointeur avant de pouvoir l'utiliser :

```
int * p_int = &i;
```

Exemple 8: Référencement d'un pointeur

Pour pouvoir utiliser la valeur pointée par un pointeur, il faut déréférencer cette valeur en utilisant « * » :

```
printf("La valeur pointée par p_int est: %i\n", *p_int); //affiche 1
```

Exemple 9: Déréférencement d'un pointeur pour accéder à la valeur pointée

7. Les tableaux

Il existe deux types de tableaux en C :

- **indexés**, qui ne contiennent qu'une colonne accessible via un index numérique ;
- **multidimensionnels**, qui contiennent plusieurs tableaux accessibles à travers leurs index respectifs.

a) Tableaux indexés d'entiers

Ces tableaux permettent de stocker des nombres. Leurs index sont représentés par des nombres et ils commencent par l'index « 0 » (zéro).

Création

Pour déclarer un tableau il suffit d'utiliser les « [ ] » pour spécifier sa taille et « { } » pour affecter un contenu :

```
int tab_int[4] = { 1, 2, 3, 4 };
```

Exemple 10: Déclaration et affectation d'un tableau d'entiers

Taille

Il est possible d'utiliser la fonction *sizeof* pour connaître la taille du tableau. Attention car la valeur retourner par *sizeof* est celle qu'occupe le tableau en mémoire. Si on reprend notre exemple précédent, le tableau contient quatre entiers, chaque entier occupe 4 octets, cela fait donc 16 octets (valeur retournée par *sizeof*).

Exemple

Si l'on souhaite connaître non pas la taille mais le nombre d'index du tableau, il faut diviser le retour de *sizeof* par la taille d'un élément.

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

int main(void) {
	int tab_int[4] = { 1, 2, 3, 4 };
	printf("La valeur de tab_int[0] est: %i\n", tab_int[0]);
	printf("La taille mémoire de tab_int est: %i\n", (int) sizeof(tab_int));
	printf("La taille d'un élément de tab_int est: %i\n", (int) sizeof(tab_int[0]));
	printf("Le nombre d'éléments de tab_int est: %i\n",(int) (sizeof(tab_int) / sizeof(tab_int[0])));
	return EXIT_SUCCESS;
}
```

Exemple 11: Création et manipulation d'un tableau d'entiers

b) Tableaux indexés de caractères (chaîne de caractères)

Ces tableaux permettent de stocker des caractères. Leurs index sont représentés par des nombres et ils commencent par l'index « 0 » (zéro) et il se termine **toujours** par le caractère terminateur de chaîne « \0 » .

Création

Prenez bien garde à toujours compter le caractère terminateur de chaîne. Sinon, toutes les fonctions qui permettent de manipuler les chaînes de caractères comme par exemple *strcpy*, *strcat* ou *strcmp* ne fonctionneront pas !

Taille

Pour déclarer un tableau il suffit d'utiliser les « [ ] » pour spécifier sa taille et « " " » pour affecter un contenu :

```
char tab_char[7] = "coucou";
```

Exemple 12: Déclaration et affectation d'un tableau de caractères

Exemple

Il est possible d'utiliser la fonction *sizeof* pour connaître la taille du tableau et cette fois-ci, à l'instar du tableau de nombres, le résultat retourné correspond bien au nombre de caractères de la chaîne (**terminateur inclus**). Si vous désirez la taille de la chaîne de caractère (**terminateur exclus**) il faut utiliser la fonction *strlen* ;

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

int main(void) {
	char tab_c1[7] = "coucou", tab_c2[3] = "le", tab_c3[6] = "monde";
	char tab_all[16] = ""; // Initialisation avec une chaîne vide (terminateur '\0' à l'index 0)
	printf("La valeur de tab_char[0] est: %c\n", tab_c1[0]);
	printf("La taille mémoire de tab_char est: %i\n", (int) sizeof(tab_c1));
	printf("La longueur de tab_char est: %i\n", (int) strlen(tab_c1));
	// Concaténation des trois chaînes
	strcat(tab_all, tab_c1); // Copie la chaîne de caractère
	strcat(tab_all, " "); // Concatène la chaîne de caractère
	strcat(tab_all, tab_c2);
	strcat(tab_all, " ");
	strcat(tab_all, tab_c3);
	printf("La chaîne concaténée est : %s\n", tab_all);
	return EXIT_SUCCESS;
}
```

Exemple 13: Création et manipulation d'un tableau de caractères

N’oubliez pas :

- d’inclure le fichier string.h pour bénéficier des fonctions qui permettent de manipuler les chaînes de caractères ;
- de toujours initialiser vos tableaux, même avec une chaîne vide ! Un tableau non initialisé ne contient pas de terminateur de chaîne et produit un résultat incertain

```
…
char tab_all[16]; // tab_all prend une valeur poubelle (garbage)
…
printf("La chaîne concaténée est : %s\n", tab_all); // Affiche → La chaîne concaténée est : ▯t▯▯▯coucou le
```

Exemple 14: Démonstration de l’utilisation d’un tableau non initialisé

Pour éviter ce phénomène, on peut toujours remplacer la **première** occurrence de *strcat* par *strcpy* qui va écraser le tableau destination par le contenu du tableau source.

Voici un récapitulatif de certaines fonctions de manipulation des tableaux de caractères :

- strcpy(s1, s2) : copie la chaîne s2 dans s1 ;
- strcat(s1, s2) : concatène la chaîne s2 à s1 ;
- strlen(s1) : donne la longueur de la chaîne s2 à s1 ;
- strcmp(s1, s2) : compare les chaînes s2 et s1 ;
- strchr(s1, ch) : retourne un pointeur sur la première occurrence de « ch » ;
- strstr(s1, s2) : retourne un pointeur sur la première occurrence de « s2 » ;

Séquences d'échappement

Il existe des chaînes de caractères qui permettent de reproduire des séquences d'échappement comme un saut à la ligne ou encore une tabulation. Ci-dessous un tableau résumant ces chaînes et leurs significations :

Séquences	Description
\n	La séquence est remplacée par le caractère ***newline*** *(saut de ligne)*
\r	La séquence est remplacée par le caractère ***carriage-return*** *(retour à la ligne)*
\t	La séquence est remplacée par le caractère tabulation
\$	La séquence est remplacée par le caractère « $ »
\"	La séquence est remplacée par le caractère « " »
\\	La séquence est remplacée par le caractère « \ »

Affichage de texte

Pour afficher du texte, on peut utiliser la fonction *printf* en spécifiant le type (format) de chacun des champs. Ci-dessous le format d’un champ :

```
%[drapeaux][taille][.précision][longueur]spécificateur
```

Exemple 15: Format d’un champ de printf

Ci-dessous un récapitulatif de quelques valeurs possibles pour les champs :

Spécificateur
c → Caractère
i / u → Entier signé et non-signé
d → Entier long signé
f → Nombre à virgule
x → Entier hexadécimal non-signé
p → Adresse de pointeur
o → Octet signé

Drapeaux
- → justification à droite
+ → ajoute le signe du nombre
# → ajoute le spécificateur pour les options *o et x* (0 et 0x)
0 → ajoute des zéros de bourrage

Taille
nombre → nombre minimum de caractères à écrire

Longueur
h → pour les entiers courts (*i*, *d*, *o*, *u* et *x*)
l → pour les entiers longs
L → pour les nombres à virgule longs

Précision
.nombre → nombre maximum de chiffres à écrire

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

int main() {
	double nombre = 125.123456789;
	short int entier = 13;
	printf("Voici un nombre avec deux chiffres après la virgule %-.3f\n", nombre);
	printf("Voici la représentation de %i en hexadécimal %#x\n", entier, entier);
	return EXIT_SUCCESS;
}
```

Exemple 1: Exemples d'utilisation des champs de printf

c) Tableaux multidimensionnels

Avec les tableaux multidimensionnels, on va tout simplement mettre un tableau dans chacun des index.

Création

Pour déclarer un tableau multidimensionnel, il suffit d'utiliser les « [ ] » deux fois de suite, sa taille et « { } » pour affecter un contenu :

```
int tab_int[2][2] = { {1, 2}, {3, 4} };
```

Exemple 16: Déclaration et affectation d'un tableau d'entiers

Taille

On peut utiliser *sizeof* pour connaître la taille en mémoire ainsi que le nombre d'éléments de chacun des index :

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

int main(void) {
	int tab_2d[2][2] ={ {1, 2}, {3, 4} };
	printf("La valeur de tab_2d[0][1] est: %i\n", tab_2d[0][1]);
	printf("La taille mémoire de tab_2d est: %i\n", (int) sizeof(tab_2d));
	printf("La nombre d'éléments de tab_2d est: %i\n", (int) (sizeof(tab_2d)/sizeof(tab_2d[0])));
	printf("La taille mémoire de tab_2d[0] est: %i\n", (int) sizeof(tab_2d[0]));
	printf("La nombre d'éléments de tab_2d[0] est: %i\n", (int) (sizeof(tab_2d[0])/sizeof(tab_2d[0][0])));
	printf("La taille mémoire de tab_2d[0][0] est: %i\n", (int) sizeof(tab_2d[0][0]));
	return EXIT_SUCCESS;
}
```

Exemple 17: Création et manipulation d'un tableau multidimensionnel d'entiers

Représentation

On peut utiliser, comme représentation intuitive, celle des matrices en lignes et colonnes :

	j = 0	j = 1
i = 0	tab_int[0][0] → 1	tab_int[0][1] → 2
i = 1	tab_int[1][0] → 3	tab_int[1][1] → 4

Cependant en mémoire, les adresses seront contiguës :

tab_int[0][0] → 1	tab_int[0][1] → 2	tab_int[1][0] → 3	tab_int[1][1] → 4

8. Les fonctions

Une fonction est une partie de code réutilisable partout dans la page et qui permet de ne pas dupliquer du code.

a) Définition

Une fonction est définie par ce qu'on appelle sa signature. Ci-dessous le schéma de création général :

```
type_retour nom_de_la_fonction(paramètre1,...,paramètreX){
	instructions
}
```

Exemple 18: Signature d'une fonction

b) Appel

Pour appeler une fonction, il suffit d'utiliser son nom suivi des « () » qui contiendront la liste des paramètres :

```
// Création de la fonction
void bonjour(char prenom[]){
        printf("Bonjour %s\s",prenom);
}
...
// Appel
bonjour("Magali"); // affichera : Bonjour Magali
```

Exemple 19: Appel d'une méthode

Dans l'exemple précédent, la fonction ne retourne rien et on l'appelle donc une méthode. **Pour retourner une valeur, il faut utiliser le mot-clé « return »**.

c) Valeur de retour

Ci-dessous, la fonction « add » qui retourne un entier :

```
int add(int num1, int num2){
        return num1+num2;
}
…
printf("La somme de 1 avec 1 est :%i\s", add(1,1));
```

Exemple 20: Appel d'une fonction

9. Les structures

Les tableaux permettent de stocker plusieurs données du même type. De manière un peu similaire, les structures permettent de définir un nouveau type de données qui peuvent stocker des types différents.

a) Déclaration

Dans l'exemple suivant, la structure Personne possède trois attributs, deux chaînes de caractères et un entier :

```
struct Personne {
        char nom[20];
        char prenom[20];
        int age;
};
```

Exemple 21: Exemple de structure

b) Initialisation

On peut maintenant utiliser cette structure dans notre code :

```
int main(void) {
	struct Personne personne;
	strcpy(personne.nom, "Dupond");
	strcpy(personne.prenom, "Antoine");
	personne.age = 21;
	accueillir(personne);
	return EXIT_SUCCESS;
}
```

Exemple 22: Initialisation d'une structure

c) Accès aux membres

Il suffit d'utiliser le '.' pour accéder aux différents attributs de cette structure :

```
void accueillir(struct Personne personne) {
	printf("Bonjour %s %s vous avez %i ans.\n", personne.prenom, personne.nom,
			personne.age);
}
```

Exemple 23: Accès aux membres d'une structure

d) Pointeurs et structures

Il est possible d'utiliser des pointeurs pour accéder aux structures. Cependant, l'accès aux membres ne se fera plus avec un '.' mais avec '→'. Dans l'exemple suivant, la fonction « initializePersonne » permet d'initialiser tous les membres de la structure :

```
void initializePersonne(struct Personne * personne, char nom[], char prenom[],
		int age) {
	strcpy(personne->nom, nom);
	strcpy(personne->prenom, prenom);
	personne->age = age;
}

int main(void) {
	struct Personne personne;
	initializePersonne(&personne, "Dupond", "Antoine", 21);
	accueillir(personne);
	return EXIT_SUCCESS;
}
```

Exemple 24: Utilisation d'un pointeur sur une structure

Dans l'exemple suivant, on permet à l'utilisateur de positionner lui-même les valeurs des différents membres :

```
void promptPersonne(struct Personne * personne) {
	printf("Entez un nom: ");
	scanf("%s",&(*personne).nom);
	printf("Entez un prénom: ");
	scanf("%s",&(*personne).prenom);
	printf("Entez un âge: ");
	scanf("%i",&(*personne).age);
}
```

Exemple 25: Déréférencement et pointeur sur une structure

10. Les opérateurs

Pour les exemples suivants, nous allons utiliser deux variables a et b qui valent respectivement 6 et 2.

a) Arithmétiques

Ci-dessous un tableau résumant les opérateurs arithmétiques utilisés en C :

Opérateur	Description	Exemple
+	addition	a + b = 8
-	soustraction	a – b = 4
*	multiplication	a * b = 12
/	division	a / b = 3
%	modulo	a % b = 0
++	incrément	a++ = 7
--	décrément	a-- = 5

b) Comparaisons

Ci-dessous un tableau résumant les opérateurs de comparaison utilisés en C, les résultats sont booléens (vrai / faux) :

Opérateur	Description	Exemple
==	égalité	a == b = faux
!=	différent	a != b = vrai
>	supérieur	a > b = vrai
<	inférieur	a < b = faux
>=	supérieur ou égal	a >= b = vrai
<=	inférieur ou égal	a <= b = faux

c) Logiques

Ci-dessous un tableau résumant les opérateurs de comparaison utilisés en C, les résultats sont booléens (vrai / faux) :

Opérateur	Description	Exemple
and	ET logique	(a && b) = vrai
or	OU logique	(a || b) = vrai
&&	ET logique	(a && b) = vrai
||	OU logique	(a || b) = vrai
!	Négation logique	!(a && b) = faux

d) Affectations

Ci-dessous un tableau résumant les opérateurs d'affectation utilisés en C, nous utiliserons une troisième variable c :

Opérateur	Description	Exemple
=	affectation	c = (a + b) équivaut à c = 8
+=	addition ET affectation	c += a équivaut à c = c + a
-=	soustraction ET affectation	c -= a équivaut à c = c - a
*=	multiplication ET affectation	c *= a équivaut à c = c * a
/=	division ET affectation	c /= a équivaut à c = c / a
%=	modulo ET affectation	c %= a équivaut à c = c % a

e) Conditionnel ou opérateur ternaire

L'opérateur conditionnel « ? » évalue dans un premier temps la condition donnée de manière booléenne puis, en fonction du résultat, choisit entre deux instructions :

- ((a > b) ? 10 : 20) équivaut à 10 ;
- ((a < b) ? 10 : 20) équivaut à 20.

11. Boucles conditionnelles et itératives

a) If

La boucle « if » permet de faire des choix entre deux possibilités ou plus.

If

C'est la forme la plus simple de créer une instruction conditionnelle :

```
if (condition){
        Instructions à exécuter si la condition est vérifiée
}
```

Exemple 26: Boucle if

If / else

Cette boucle permet un contrôle plus fin en autorisant l'exécution d'une instruction si la condition n'est pas vérifiée :

```
int age = 20 ;
if (age > 18){
        puts("Vous êtes apte à conduire !");
}else{
        puts("Patientez encore un peu avant de conduire...");
}
```

Exemple 27: Boucle if / else

If / else if

C'est la forme la plus évoluée de boucle pour créer des expressions conditionnelles. Elle offre une très grande souplesse en permettant la gestion de choix multiples :

```
int age = 20 ;
if (age > 16){
        puts("Vous êtes apte à conduire un scooter !");
}else if(age > 18){
        puts("Vous êtes apte à conduire une voiture !");
}else{
        puts("Patientez encore un peu avant de conduire...");
}
```

Exemple 28: Boucle if / else if

b) For

La boucle « for » permet de répéter une instruction. Elle se décompose en trois parties :

```
for (initialisation; condition / test; instruction itérative){
        Instructions
}
```

Exemple 29: Schéma d'une boucle « for »

- L'initialisation permet de mettre en place une variable qui fait office de compteur (eg. « int i = 0 ») ;
- la condition ou test s'il échoue permet d'arrêter l'itération (eg. i < 5) ;
- l'instruction itérative permet d'incrémenter ou décrémenter le compteur (eg. i++) ;

Ci-dessous un exemple de boucle « for » :

```
int nombres[] = { 1, 2, 3 };
int count = sizeof(nombres) / sizeof(nombres[0]);
int i;
for (i = 0; i < count; i++) {
        printf("La valeur de nombres[%i] est %i\n", i, nombres[i]);
}
```

Exemple 30: Boucle « for »

c) While

La boucle « while » est le moyen le plus simple d'itérer. Elle suit le schéma suivant :

```
while (condition){
    Instructions
}
```

Exemple 31: Schéma de la boucle « while »

Le but d'une telle boucle est de répéter le bloc d'instructions tant que la condition est vérifiée :

```
int nombres[] = { 1, 2, 3 };
int count = sizeof(nombres) / sizeof(nombres[0]);
int i = 0;
while(i < count){
        printf("La valeur de nombres[%i] est %i\n", i, nombres[i]);
        i++ ;
}
```

Exemple 32: Boucle « while »

d) Do...while

La boucle « do...while » permet de placer la vérification de la condition en fin de boucle, ce qui permet d'exécuter au moins une fois les instructions :

```
do{
    Instructions
}while (condition) ;
```

Exemple 33: Schéma de la boucle « do...while »

Veuillez noter le « ; » en fin de boucle.

12. Contrôle des boucles

C fournit les outils nécessaires pour contrôler les itérations car il se peut qu'il y ait une situation où il soit nécessaire de quitter une boucle sans avoir atteint la fin de l'itération.

a) Break

Le mot-clé break permet de quitter une boucle avant d'avoir atteint la fin de celle-ci. Cela placera le curseur d'exécution du code à la fin de la boucle, c'est à dire après l'accolade fermante « } ».

```
int nombres[] = { 1, 2, 3 };
int count = sizeof(nombres) / sizeof(nombres[0]);
int i;
for (i = 0; i < count; i++) {
	printf("La valeur de nombres[%i] est %i\n", i, nombres[i]);
	if(nombres[i] == 2){
		break;
	}
}
```

Exemple 34: Utilisation de « break »

b) Continue

Le mot clé « continue » permet de « sauter » une itération en passant directement à la suivante :

```
int nombres[] = { 1, 2, 3 };
int count = sizeof(nombres) / sizeof(nombres[0]);
int i;
for (i = 0; i < count; i++) {
	if(nombres[i] == 1){
		continue;
	}
	printf("La valeur de nombres[%i] est %i\n", i, nombres[i]);
}
```

Exemple 35: Utilisation de « continue »

13. Best practices

Chaque entreprise possède ses propres standards de programmation. Cependant, il existe une base essentielle et commune à tous : **les bonnes pratiques** !
Voici quelques raisons d'utiliser de telles pratiques :

- La majeure partie du temps, vous êtes amenés à travailler en binômes et il est important que votre homologue puisse relire le code que vous produisez sans avoir à le déchiffrer ;
- La simplicité et la clarté qui émanent d'un code consistent permettent d'éviter les erreurs communes ;
- Si vous relisez votre code après un certain temps, ce code vous paraît simple ;
- Cela fait partie des standards de l'industrie de suivre les bonnes pratiques car elles garantissent une certaine qualité.

Il existe certaines lignes directrices qu'il est intéressant de suivre lorsque l'on code en C :

- **Indentation**

  Utilisez une indentation de quatre espaces et surtout pas de tabulation car elles ont un rendu différent selon les systèmes.

- **Longueur de ligne**

  Une longueur de ligne de 75 à 85 caractères permet une bonne lisibilité.

- **Structure de contrôle**

  Cela concerne toutes les boucles (if, for, while, ...). Les contrôles de structure doivent avoir un espace entre le mot de contrôle et la première parenthèse, pour les différencier des appels de fonctions. Il est fortement recommandé d'utiliser les accolades même quand, techniquement, elles ne sont pas nécessaires.

```
if ((condition1) || (condition2)) {
    action1;
} elseif ((condition3) && (condition4)) {
    action2;
} else {
    action par défaut;
}
```

Exemple 36: Syntaxe d'une structure de contrôle

- **Utilisation du standard C99**

  Cela permet, entre autre, d'utiliser certains « raccourcis » :

```
Sans C99 :
int i ;
for(i = 0 ; i < 10 ; i++){
        …
}
Avec C99 :
for(int i = 0 ; i < 10 ; i++){
        …
}
```

Exemple 37: Boucle for en C99

  Il est bien d'utiliser les types spécifiques pour les entiers (uint8_t, int32_t, …) ;

- **Appel de fonctions**

  Les fonctions doivent être appelées avec aucun espace entre le nom de la fonction et la parenthèse ouvrante, un espace entre virgules et paramètres, et aucun espace entre le dernier paramètre et la dernière parenthèse.

```
int var = function($arg1, $arg2, $arg3);
```

Exemple 38: Syntaxe d'un appel de fonction

Il existe encore bien des points qui méritent d'être abordés mais les principaux sont exposés. L'objectif d'un standard est de vous permettre de produire un code de qualité et compréhensible par le plus grand nombre.

14. Index des exemples

Index des exemples