




## – Algorithmique – Travaux Pratiques – Série N°1

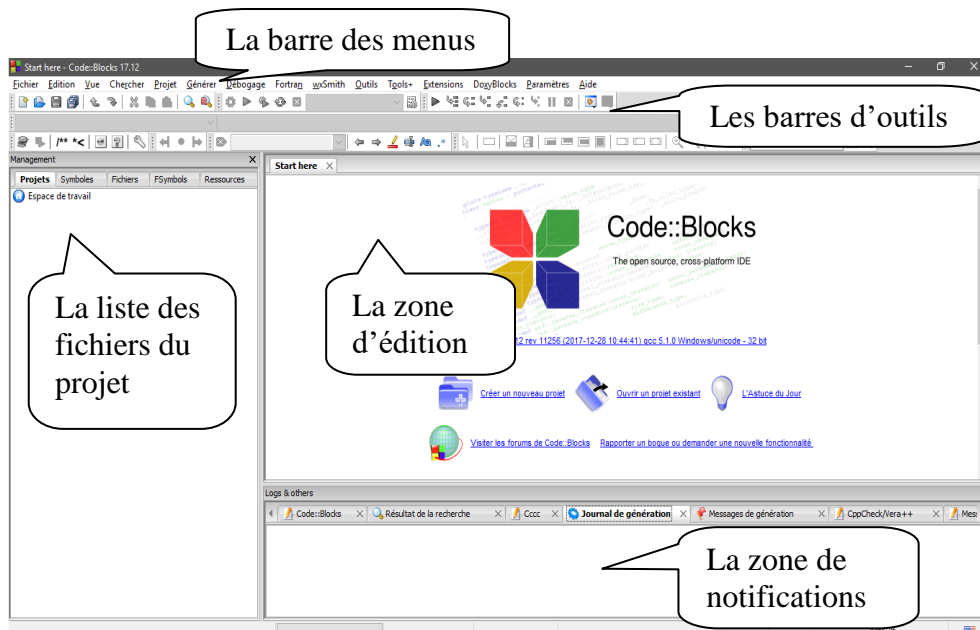
**Introduction** : Pour programmer en langage C, il existe plusieurs IDE (Environnement de Développement Intégré). On peut citer **Code::Blocks**, **DevC++**, **Eclipse**, **Visual Studio express**.

Un IDE est un ensemble d'outils qui inclut un **éditeur de texte**, un **compilateur** et un **débogueur**. L'éditeur de texte permet d'écrire le programme source. Le compilateur vérifie si le programme écrit respecte la syntaxe du langage de programmation utilisé puis génère le programme exécutable. Le débogueur facilite la recherche d'erreurs de logique dans le programme. Il permet notamment l'exécution du programme pas-à-pas (instruction par instruction), l'affichage des valeurs des variables à tout moment, ... Nous choisissons de travailler avec **Code::Blocks**, les programmes écrits marchent parfaitement avec les autres IDEs.

### Exercice 1 : Mon premier programme en langage C

Écrire un programme qui affiche « Bonjour tout le monde ! ».

**ETAPE 1** – Lancer le logiciel Code::Blocks en cliquant sur l'icône  affichée sur le bureau de Windows. Vous apercevez la fenêtre de l'IDE suivante :



**ETAPE 2** – Pour créer un programme C avec Code :Blocks, on peut procéder de différentes manières. On vous propose deux méthodes. Vous pouvez choisir l'une ou l'autre.

#### 2.1- Méthode 1 – Création automatique

- Cliquer sur le **menu Fichier → Nouveau**
- **Projet...**
- Choisir "**Console Application**".

- Cliquer sur "**next**" (≡ Suivant).
- Choisir "**C**" (c'est-à-dire le langage C).
- Cliquer sur "**next**".

- Donner un nom au projet (**Projet1** par exemple) et choisir le répertoire où il sera créé (le Bureau de Windows par exemple).
- Cliquer sur **"next"**.
- Ne rien changer sur cette page.
- Cliquer sur **"Finish"**.

*Dans la partie gauche de l'interface, un projet "**Projet1**" est créé.*

- Double Cliquer sur **"Sources"**, vous trouverez un fichier **"main.c"** qui est créé.
- Double cliquer sur **"main.c"**, il s'affiche dans la zone d'édition (partie droite).
- A la ligne 6 du programme source, remplacer le message **"Hello world!\n"** par **"Bonjour tout le monde !\n"**
- On peut aussi supprimer la ligne 2, car on n'en a pas besoin pour le moment.

## 2.2- Méthode 2 – Création manuelle

- Cliquer sur le **menu Fichier → Nouveau → Fichier...**
- Choisir **"C/C++ source"**

- Cliquer sur **"next"**
- Choisir **"C"** (c'est-à-dire le langage C)
- Cliquer sur **"next"**
- Donner un nom au programme C (**"main.c"** par exemple) et choisir le répertoire où il sera créé.
- Cliquer sur **"Finish"**

*Dans la partie droite de l'interface, un fichier vide au nom "**main.c**" est créé.*

- Dans la partie de droite, écrire le code suivant :

```
#include <stdio.h>

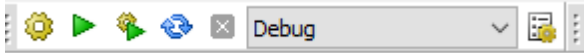
int main()
{
    printf("Bonjour tout le monde !\n");
    return 0;
}
```

- Enregistrer le programme source en cliquant sur le **menu Fichier → Enregistrer le fichier** (ou avec le clavier appuyer sur **Ctrl+S** ou **Alt+F** puis choisir **Enregistrer le fichier**).

## ETAPE 3 – Compiler le programme source : - Cliquer sur le **menu Générer → Générer et exécuter**.

*Vous obtenez une console (fenêtre noire) qui affiche le message "**Bonjour tout le monde !**".*

*Pour fermer cette fenêtre appuyer sur n'importe quelle touche du clavier ou avec la souris cliquer sur l'icône **X** en haut à droite de la fenêtre.*

**Remarque** : Les commandes des menus les plus utilisées (Enregistrer, Générer, ...) sont accessibles par des barres d'outils. Par exemple, la barre d'outils  permet d'accéder aux commandes : Générer, Exécuter, Générer et exécuter, Régénérer, ...

**Exercice 2** : Écrire un programme qui calcule la somme de deux nombres entiers **a** et **b**.

**Exercice 3** : Modifier le programme écrit dans l'exercice 2 pour qu'il calcule aussi :

- 1) La différence entre **a** et **b**.
- 2) La multiplication de **a** par **b**.
- 3) La division de **a** par **b**.
- 4) Le reste de division de **a** par **b** (**a % b**). Dans 3) et 4) Il faut que **b** ne soit pas nul.

**Exercice 4** : Écrire un programme qui permet de permuter les valeurs de deux entiers **a** et **b**.

# Programmer en langage C avec Code::Blocks

## I – Quelques commandes de l'environnement Code::Blocks

Le menu **Fichier** :

- La commande **Nouveau** permet de créer un nouveau fichier source vide, un nouveau projet, ...
- La commande **Ouvrir** permet de charger un programme à partir du disque dur (ou une clé USB).
- La commande **Enregistrer** (**Ctrl+S**) permet de sauvegarder un programme sur le disque dur.
- La commande **Enregistrer sous...** permet de sauvegarder un programme sur le disque dur en lui donnant un nouveau nom.
- La commande **Quitter** (**Ctrl+Q**) permet de quitter Code::Blocks. N'oubliez pas d'enregistrer votre programme s'il a été modifié.

Le menu **Edition** :

- La commande **Défaire (Annuler)** (**Ctrl+Z**) permet d'annuler la dernière modification apportée au programme. On peut annuler plusieurs modifications en appuyant plusieurs fois sur **Ctrl+Z**.
- La commande **Refaire** (**Ctrl+Shift+Z**) permet de rétablir la dernière modification apportée au programme. On peut rétablir plusieurs modifications en appuyant plusieurs fois sur **Ctrl+Shift+Z**.
- La commande **Couper** (**Ctrl+X**) permet d'effacer une partie du programme afin de la déplacer vers un autre endroit dans le programme. Pour sélectionner une partie du programme avec le clavier, appuyer sur la touche **Shift** ( **⇧** ) et utiliser les touches de direction.
- La commande **Copier** (**Ctrl+C**) permet de copier une partie du programme afin de la dupliquer dans un autre endroit dans le programme.
- La commande **Coller** (**Ctrl+V**) permet d'insérer une partie du programme à l'endroit où se trouve le curseur. Cette partie du programme a été déjà sélectionnée auparavant et l'une des deux commandes **Couper** ou **Copier** a été déjà utilisée.

**Astuce** : Le raccourci clavier **Ctrl+D** permet de dupliquer la ligne où se trouve le curseur.

Pour dupliquer plusieurs lignes, il faut les sélectionner puis appuyer sur **Ctrl+D**.

Le menu **Générer (Compiler)** :

- La commande **Générer (Compiler)** (**Ctrl+F9**) permet de compiler le programme afin de vérifier s'il contient des erreurs. Si c'est le cas, des messages d'erreurs (et avertissements) vont s'afficher dans la zone de notifications. Si le programme ne contient aucune erreur, le message "Génération terminée" s'affiche ; Cela signifie que le programme exécutable est généré et que vous pouvez l'exécuter.
- La commande **Exécuter** (**Ctrl+F10**) permet d'exécuter le programme s'il a été généré. Si le programme a été modifié, il faut le générer de nouveau afin de prendre en compte les dernières modifications. Sinon, c'est l'ancien programme (avant modification) qui s'exécutera !
- La commande **Générer et exécuter** (**F9**) permet de compiler et exécuter le programme.

## II - Règles générales du langage C

- 1- En langage C, chaque instruction ou déclaration se termine par un point-virgule ‘;’.
- 2- Le langage C est sensible à la casse ; c’est-à-dire qu’il fait la différence entre majuscules et minuscules. Un nom contenant des majuscules est différent du même nom écrit en minuscules.

**Exemple :** Le nom **varx** est différent de **varX**, **VarX** ou **VARX**.

- 3- Les mots clés du langage C sont : *auto, break, case, char, const, continue, default, do, double, else, enum, extern, float, for, goto, if, inline, int, long, return, short, signed, sizeof, static, struct, switch, typedef, union, unsigned, void, volatile, while.*

## III – Transformation de la syntaxe algorithmique vers le langage C

### III.1 – Traduction d’un algorithme simple vers le langage C

<b>Algorithme Permutation</b>	01	<i>/* Programme Permutation. Un commentaire multilignes.</i>
	02	<i>Ci-dessous la directive d’inclusion de la bibliothèque</i>
	03	<i>stdio où les fonctions scanf et printf sont définies */</i>
<b>Var A, B : Entier</b>	04	<b>#include &lt;stdio.h&gt;</b> <i>// Ajouter la bibliothèque stdio</i>
	05	
	06	<b>int main( )</b> <i>// fonction main = programme principal.</i>
<b>Début</b>	07	{
	08	<b>int a, b ;</b> <i>//Commentaire jusqu’à la fin de la ligne.</i>
	09	
Lire(A)	10	<b>printf("Donner la valeur de A. A= ? ") ;</b>
	11	<b>scanf("%d", &amp;a) ;</b>
Lire(B)	12	<b>printf("Donner la valeur de B. B= ? ") ;</b>
	13	<b>scanf("%d", &amp;b) ;</b>
	14	<b>printf("Valeurs initiales A=%d et B=%d\n", a, b) ;</b>
A ← A + B	15	<b>a = a + b ;</b> <i>// On peut l’écrire aussi a += b ;</i>
B ← A – B	16	<b>b = a - b ;</b>
A ← A – B	17	<b>a = a - b ;</b> <i>// On peut l’écrire a -= b ;</i>
Ecrire(A, B)	18	<b>printf("Valeurs finales A=%d et B=%d\n", a, b) ;</b>
	19	
	20	<b>return 0 ;</b>
<b>Fin</b>	21	}

Un programme C a la structure suivante :

- ✓ Partie 1 : Les directives du préprocesseur. Les lignes commençants par le caractère ‘#’ (voir ligne 4).
- ✓ Partie 2 : Le programme principal (fonction int main) (voir ligne 6). Il contient des déclarations (voir ligne 8) et des instructions (de la ligne 10 à la ligne 20).
- ✓ Un programme peut contenir des commentaires (lignes 1 à 3 et lignes 6, 8, 15 et 17). Un commentaire est une partie du fichier source qui a pour but d’expliquer le fonctionnement du programme sans que le compilateur ne la prenne en compte. Il existe deux types de commentaire :
  - Type 1 : Peut s’étaler sur plusieurs lignes et il a la syntaxe suivante : **/\* commentaire trop long \*/**.
  - Type 2 : Commentaire court, il commence par ‘//’ et se termine à la fin de ligne (Lignes 4, 6, 8, 15, 17).

### III.2 – Les types de données

A l'inverse du langage algorithmique où les types standards ont une taille illimitée, en langage C, les types de données ont une taille limitée car dépendante de l'architecture de l'ordinateur.

On peut traduire les types standards en algorithmique vers le langage C selon le tableau suivant :

En Algorithmique	Caractère	Entier	Réel	Chaîne (de caractères)	Booléen
En langage C	char	int	float	Tableau de caractères	N'existe pas par défaut

✓ En langage C, un caractère est une variable de type **char** qui prend 1 octet (= 8 bits) en mémoire. La table ASCII (figure ci-dessous) est un tableau de 256 caractères, numérotés de 0 à 255, où les 32 premiers sont des caractères non affichables et tous les autres sont affichables : lettres, chiffres, ponctuations, symboles, ...

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US	
32	SP	!	"	#	\$	%	&	'	(	)	*	+	,	-	.	/	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?	
64	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_	
96	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	{		}	~	DEL	
128	Ç	ü	é	â	ä	à	å	ç	ê	ë	è	ï	î	ï	Ä	Å	É	æ	Æ	ô	ö	û	ü	ý	ÿ	Ö	Ü	ø	£	Ø	×	f	
160	á	í	ó	ú	ñ	Ñ	ª	º	¿	®	¬	½	¼	¿	«	»	¡	í	í	í	í	í	í	í	í	í	í	í	í	í	í	í	í
192	+	-	-	+	-	+	ä	Ä	+	+	-	-	-	-	+	±	ð	Ð	Ê	Ê	Ê	Ê	Ê	Ê	Ê	Ê	Ê	Ê	Ê	Ê	Ê	Ê	
224	Ó	ß	Ô	Õ	Ö	Ö	µ	þ	þ	Ú	Û	Ü	Ý	Ý	-	-	-	±	=	¼	¶	§	÷	÷	÷	÷	÷	÷	÷	÷	÷	÷	

**Astuce** : Pour écrire un caractère qui ne figure pas sur le clavier, on maintient la touche **Alt** du clavier appuyée puis on tape le code du caractère. **Exemple** : pour écrire le caractère '¥', utiliser **Alt+190**.

✓ En langage C, le type **char** est un cas particulier du type entier, on peut travailler avec le caractère ou son code ASCII indifféremment. Si **car** est une variable de type **char**, on peut écrire **car = 'A'** ou **car = 65**.

✓ Le type **int** prend, en général, 4 octets en mémoire, et permet d'utiliser des nombres entiers compris entre -2147483648 ( $-2^{31}$ ) et +2147483647 ( $+2^{31} - 1$ ). S'il y a besoin de nombre plus grands, le type **long int** (ou tout simplement **long**) est utilisé. Si le type **long** n'est pas suffisant, le type **long long int** (ou tout simplement **long long**) est utilisé.

✓ S'il y a besoin de petits entiers, le type **short** ([-32768 .. +32768]) ou **char** ([-128 .. +127]) est utilisé.

✓ Si seulement les nombres positifs sont nécessaires, les types **char**, **short**, **int**, **long** ou **long long** sont précédés par le mot **unsigned** (non-signé), on obtient par exemple le type **unsigned int**.

✓ Le type **float** prend, en général, 4 octets en mémoire, et permet d'utiliser des nombres réels (positifs ou négatifs) compris entre  $3.4 \times 10^{-38}$  et  $3.4 \times 10^{38}$  avec une précision de 6 chiffres après la virgule (simple précision). S'il y a besoin de plus de précision, le type **double** (précision double : de 12 chiffres après la virgule) est utilisé. Si le type double n'est pas suffisant, le type **long double** (précision étendue) est utilisé avec la garantie que : précision étendue  $\geq$  précision double.

✓ Le langage C ne possède pas de type booléen dédié. La valeur zéro est considérée comme **FAUX**.

Toute autre valeur différente de zéro est considérée comme **VRAIE**. Cela signifie que n'importe quelle expression peut être utilisée à l'intérieur des tests (à la place des conditions).

**Remarque importante** : Dans nos programmes C, le type **int** est utilisé à la place du type **Entier** utilisé dans l'écriture des algorithmes et le type **float** est utilisé à la place du type **Réel**.

### III.3 – Un identificateur (identifiant)

Un identificateur est un nom (ou un mot) constitué d'une suite de lettres de l'alphabet ('A'..'Z' et 'a'..'z') et de chiffres ('0'..'9') **commençant par une lettre** (il ne peut pas commencer par un chiffre).

Le caractère '\_' souligné (underscore) est considéré comme une lettre.

**Remarques** : ✓ L'identificateur doit être différent de tous les mots clés (char, int, float, if, while, for, ...).

✓ Il ne doit pas contenir des espaces. L'espace est, en général, remplacé par le caractère '\_' souligné.

✓ L'identificateur ne doit pas contenir des caractères accentués : é, ê, è, ù, à, î, ç, ...

### III.4 – Déclaration de variables

✓ Une variable est déclarée avec la syntaxe suivante : **type nomVariable ;**

✓ **type** est l'un des types définis en langage C ( char, int, float ...).

✓ **nomVariable** est un identificateur.

✓ Une variable peut être initialisée à la déclaration : **type nomVariable = valeurInitiale ;**

✓ Plusieurs variables d'un même type peuvent être déclarées (et initialisées) dans la même déclaration.

**Syntaxe** : **type nomVariable1 [= valeur1], nomVariable2 [= valeur2] ... nomVariableN [= valeurN] ;**

**Exemples** : char a ;

int n=5 , nEtdudiant ;

float x1, x2, delta=0 ;

### III.5 – Déclaration de constantes

✓ Une constante est déclarée avec la syntaxe suivante : **const type NOM\_CONSTANTE = Valeur ;**

✓ **type** est l'un des types définis en langage C ( char, int, float ...).

✓ **NOM\_CONSTANTE** est un identificateur.

**Exemples** : const char CAR = 'a' ; const char CAR\_A\_MAJ = 65 ;

const int NB\_MODULES = 8 ; const float PI = 3.14f ; // Ajouter **f** pour float

const double VIT\_LUMIERE = 3E8 ; // Vitesse de la lumière **3x10<sup>8</sup>** m/s. Notation exponentielle

**Remarque** : Par convention, le nom d'une variable est écrit en minuscules (ou au moins commence par une minuscule). Le nom d'une constante est écrit uniquement en majuscules.

### III.6 – L'instruction de lecture : scanf

*scanf* est une fonction de la bibliothèque *stdio*. Elle permet de donner une valeur à une variable à partir du clavier. Il faut que la variable utilisée soit déclarée précédemment.

**Syntaxe** : **scanf("%format", &nomVariable) ;**

Les formats les plus utilisés sont :

Type	char	int	long, long long	float	double, long double	Chaîne
Format	c	d (signé) u (non signé)	ld ou lu (non signé)	f ou e ou g	lf ou le ou lg	s

✓ On peut regrouper plusieurs lectures dans la même instruction scanf. Pour lire deux variables on écrit :

**scanf("%format1 %format2", &nomVariable1, &nomVariable2) ;**

### III.7 – L’instruction d’écriture : printf

**printf** est une fonction de la bibliothèque *stdio*. Elle permet l’affichage à l’écran.

Il existe deux types d’affichage :

1- Affichage d’un texte : **Syntaxe** : **printf**("texte") ; **Exemple** : **printf**("Donner la valeur de a ") ;

2- Affichage de la valeur d’une variable : **Syntaxe** : **printf**("%format", nomVariable) ;

**Exemple** : Si *a* est une variable de type entier, on peut écrire : **printf**("%d", a) ;

Les formats les plus utilisés sont :

Type	char	int	long, long long	float, double	long double	Chaîne
Format	c	d (signé), u (non signé)	ld (signé), lu (non signé)	f ou e ou g	lf ou le ou lg	s

✓ On peut regrouper plusieurs écritures dans la même instruction printf. Si on a  $s = a + b$ , on peut, par exemple écrire : **printf**("La somme de %d et %d = %d\n", a, b, s) ;

✓ Certains caractères spéciaux ont une signification particulière pour printf :

\n : Saut de ligne                                  \t : Tabulation                                  \\ : Affiche le caractère \

%% : Affiche le caractère %                      \' : Affiche le caractère '                      \" : Affiche le caractère "

### III.8 – Opérateurs arithmétiques et logiques (booléens)

Algorithmique	+ - *	mod	<b>div</b>	/	Non (Logique)	Et (Logique)	Ou (Logique)
Langage C	+ - *	%	/	!	&& (2 fois &)	(2 fois AltGr+6)	

**Remarque** : L’opérateur / est le même pour la division entière et la division réelle. Si les deux opérandes (facteurs) sont entiers, c’est la division entière qui est appliquée. Si au moins l’un des deux facteurs est réel, alors c’est la division réelle qui est appliquée. **Exemple** :  $7/2 = 3$  mais  $7.0/2 = 7/2.0 = 7.0/2.0 = 3.5$

### III.9 – Opérateurs de comparaison (de relation)

Algorithmique	<	≤	=	≠	≥	>
Langage C	<	<=	== (2 fois le symbole =, à ne pas confondre avec l’affectation)	!=	>=	>

### III.10 – L’instruction d’affectation

Algorithmique	←	$y \leftarrow x + 17$ (exemple)
Langage C	=	$y = x + 17$

**Cas particuliers (affectations composées) :**

Soit OP l’un des opérateurs arithmétiques suivants : +, -, \*, /, %.

✓ Si l’instruction d’affectation est de la forme **Variable1 = Variable1 OP expression**, alors l’écriture peut être allégée comme suit : **Variable1 OP= expression**. Il ne doit y avoir aucun espace entre OP et =.

**Exemple** : L’instruction  $x = x + y * z$ , peut être écrite tout simplement  $x += y * z$ .

✓ Si l'instruction d'affectation est de la forme **Variable1 += 1** (ou **Variable1 = Variable1 + 1**), alors l'écriture peut être allégée comme suit : **Variable1++** (On appelle ++ opérateur d'incrément).

**Exemple** : L'instruction **x += 1** (ou **x = x + 1**) peut être écrite tout simplement **x++**.

✓ Si l'instruction d'affectation est de la forme **Variable1 -= 1** (ou **Variable1 = Variable1 - 1**), alors l'écriture peut être allégée comme suit : **Variable1--** (On appelle -- opérateur de décrément).

**Exemple** : L'instruction **x -= 1** (ou **x = x - 1**) peut être écrite tout simplement **x--**.

**Résumé** : L'affectation a, en tout, les opérateurs suivants : =, +=, -=, \*=, /=, %=, ++, --.

### **III.11 – Règles de priorité des opérateurs**

Les opérateurs n'ont pas la même priorité. Ci-dessous les opérateurs sont classés par ordre décroissant de priorité. Si plusieurs opérateurs de même priorité existent dans une expression, ces opérateurs sont associatifs à gauche. C'est-à-dire l'opération la plus à gauche est évaluée (exécutée) en premier.

**Remarque importante** : En cas de doute, il faut utiliser des parenthèses.

- 1- ( ) les parenthèses
- 2- !(Non) ++(incrément) --(décrément) -( moins unaire) +(plus unaire) (**cast**)
- 3- \* / %
- 4- +(plus binaire) -(moins binaire)
- 5- < <= > >=
- 6- ==(égal) !=(différent)
- 7- && (ET logique)
- 8- || (OU logique)
- 9- = += -= \*= /= %= (Affectation)

### **IV – Correction De Quelques Erreurs**

- 1- L'erreur "**error: expected ';' before ...**" est très fréquente. Elle signifie que le point-virgule est manquant à l'endroit indiqué par le compilateur.
- 2- L'erreur "**error: 'varx' undeclared (first use in this function)**" est aussi très fréquente. Elle signifie que la variable 'varx' n'est pas déclarée. Parfois, la variable 'varx' est déclarée mais avec une autre écriture (par exemple 'VarX') car le langage C fait la différence entre les minuscules et les majuscules.
- 3- L'affectation utilise un seul symbole = et l'opérateur d'égalité deux symboles == ce qui donne ==. L'avertissement "**warning: suggest parentheses around assignment used as truth value**" signifie qu'une affectation est utilisée à la place d'une condition d'égalité.
- 4- L'erreur "**error: lvalue required as left operand of assignment**" est rencontrée quand une affectation ne trouve pas une variable à gauche. Cette erreur est rencontrée, en général, quand il y a une expression à gauche de l'opérateur de l'affectation. Si, par erreur, on a écrit l'expression suivante :  
 $x+y = z + 5$ . La syntaxe correcte est  $x+y == z + 5$ .