Université A. Mira Béjaia
Faculté de Sciences Exactes
Département Mathématique
L1 Math (tronc commun)
Module : ASD1

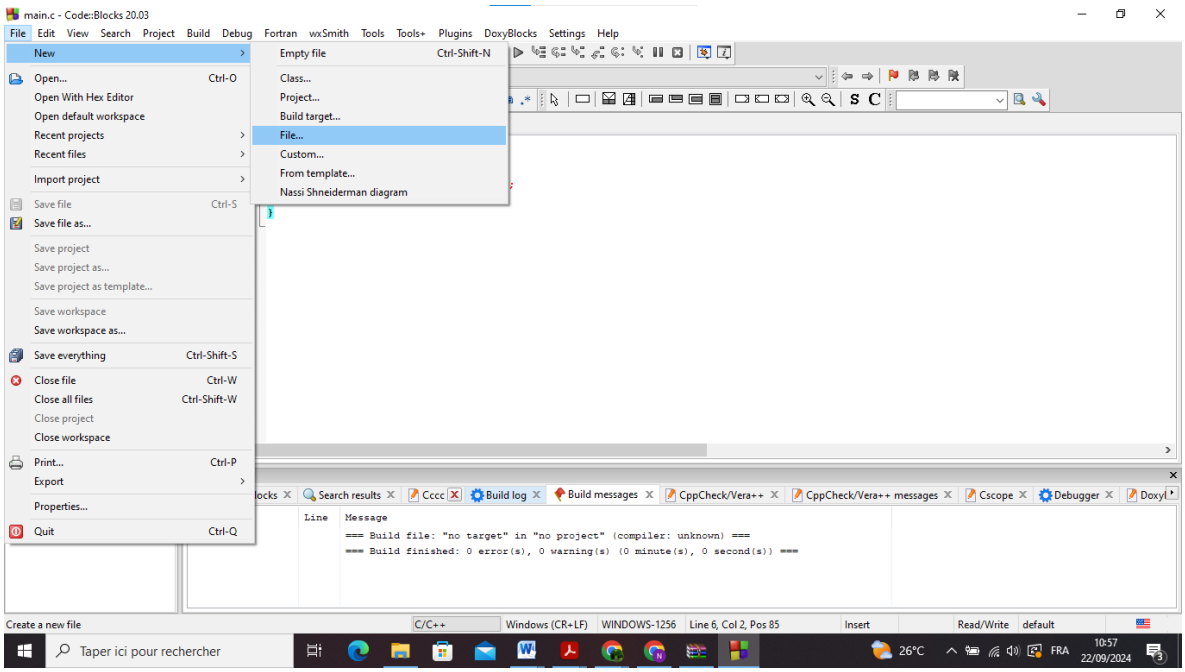
Support de TP : Initiation au langage C

1) Mon premier programme en langage C

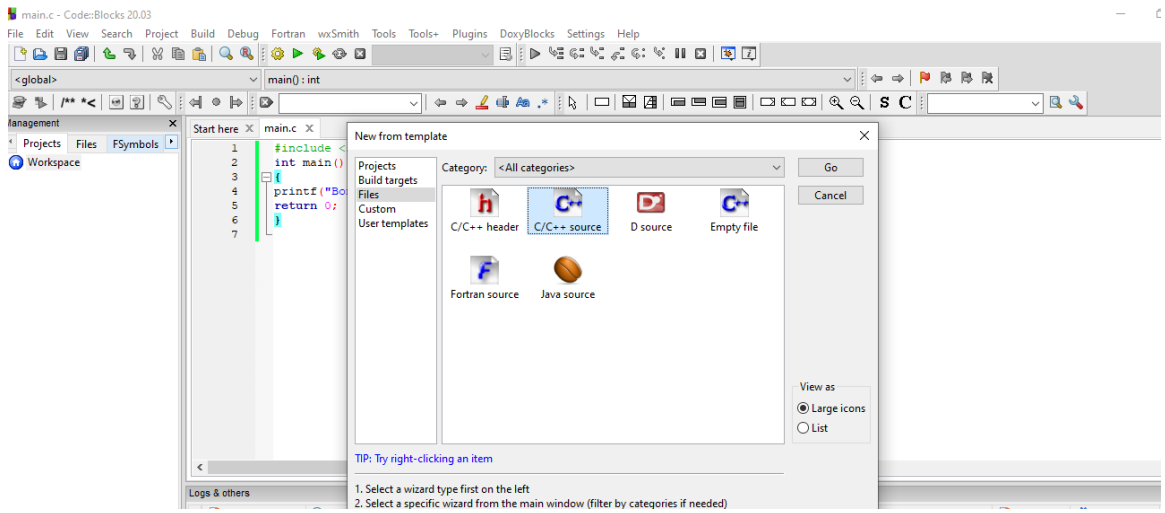
Écrire un programme qui affiche « Bonjour tout le monde ! ».

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.



- Cliquer sur le menu File → New → File...
- 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 :

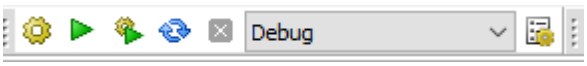
```
#include <stdio.h>
int main()
{
printf("Bonjour tout le monde !\n");
return 0;
}
```

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

ETAPE 3 : Compiler le programme source : - Cliquer sur le menu **Build → Bluid and run.**

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.

Les commandes des menus les plus utilisées (**Save, Build, ...)** sont accessibles par des barres d'outils. Par exemple**, la barre d'outils** permet d'accéder aux commandes : **Build, Run, Bluid and run …**



2) Quelques commandes de l'environnement Code::Blocks

- Le menu ***File*** :
  - La commande ***New*** permet de créer un nouveau fichier source vide, un nouveau projet, …
  - La commande ***Open*** permet de charger un programme à partir du disque dur (ou une clé USB).
  - La commande ***Save file (Ctrl+S)*** permet de sauvegarder un programme sur le disque dur.
  - La commande ***Save file as…*** permet de sauvegarder un programme sur le disque dur en lui donnant un nouveau nom.
  - La commande ***Quit (Ctrl+Q)*** permet de quitter Code::Blocks. N'oubliez pas d'enregistrer votre programme s'il a été modifié.
- Le menu ***Edit*** :
  - La commande ***Undo (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 ***Redo (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 ***Cut (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 ***Copy (Ctrl+C)*** permet de copier une partie du programme afin de la dupliquer dans un autre endroit dans le programme.

- La commande ***Paste (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 ***Cut*** ou ***Copy*** 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 ***Build (Compiler)*** :
  - La commande ***Build (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 ***Run (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 ***Build et run (F9)*** permet de compiler et exécuter le programme.

3) 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.

4) Transformation de la syntaxe algorithmique vers le langage C

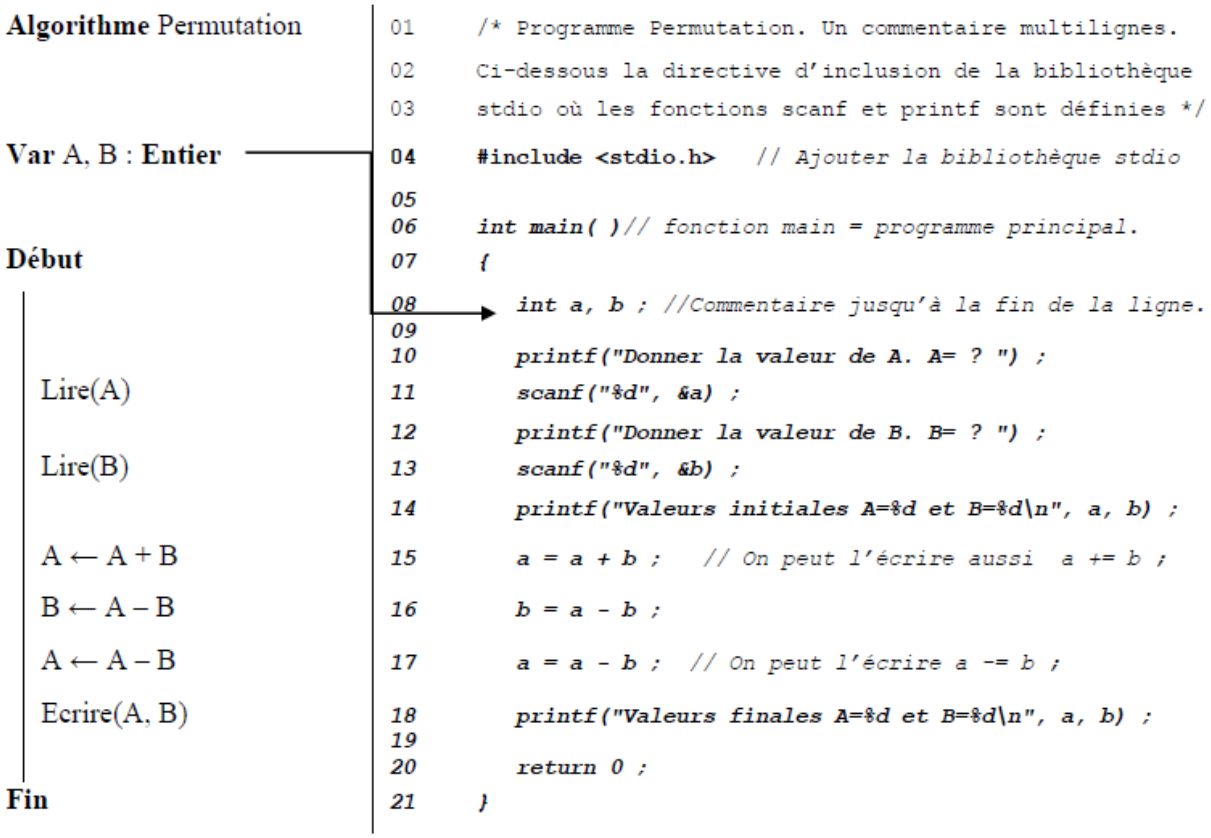
- **Traduction d'un algorithme simple vers le langage C**

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).



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	ì	Ä	Å	É	æ	Æ	ô	ö	ò	û	ù	ÿ	Ö	Ü	ø	£	Ø	×	ƒ
160	á	í	ó	ú	ñ	Ñ	ª	º	¿	®	¬	½	¼	¡	«	»	¦	¦	¦	¦	¦	Á	Â	À	©	¦	¦	+	+	¢	¥	+
192	+	-	-	+	-	+	ã	Ã	+	+	-	-	¦	-	+	¤	ð	Ð	Ê	Ë	È	ı	Í	Î	Ï	+	+	¦	_	¦	Ì	¯
224	Ó	ß	Ô	Ò	õ	Õ	µ	þ	Þ	Ú	Û	Ù	ý	Ý	¯	´	-	±	=	¾	¶	§	÷	¸	°	¨	·	¹	³	²	¦	

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.

Remarque :

- 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, ï, ç, …

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 C;

int n=2 , nbr ;

float x1, x2, taux=0.5 ;

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 int NB_MODULES = 8 ; const **float PI = 3.14f** ; // Ajouter f pour float

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.

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 (signé), 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) ;**

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 '

Opérateurs arithmétiques et logiques (booléens)

Algorithmique	+ - *	mod	**div**	/	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**

Opérateurs de comparaison (de relation)

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

L'instruction d'affectation

Algorithmique	←	y ← 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 **Variable = Variable1 OP expression**, alors l'écriture peut être allégée comme suit : **Variable 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 **Variable += 1** (ou **Variale = Variable + 1**), alors l'écriture peut être allégée comme suit : **Variable++** (On appelle **++ opérateur d'incrémentation**). **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 Variale1 = Variable1 - 1), alors l'écriture peut être allégée comme suit : **Variable--** (On appelle **-- opérateur de décrémentation**). **Exemple :** L'instruction x -= 1 (ou x = x - 1) peut être écrite tout simplement x--.

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

Correction De Quelques Erreurs

- L'erreur "**error: expected ';' before …"** est très fréquente. Elle signifie que le point-virgule est manquant à l'endroit indiqué par le compilateur.
- 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.
- 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.
- 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.