Langage C
2023-2024

Eductive

CAMPUS
EDUCTIVE
AIX-EN-PROVENCE

Campus Eductive – Aix en Provence
ESGI 1

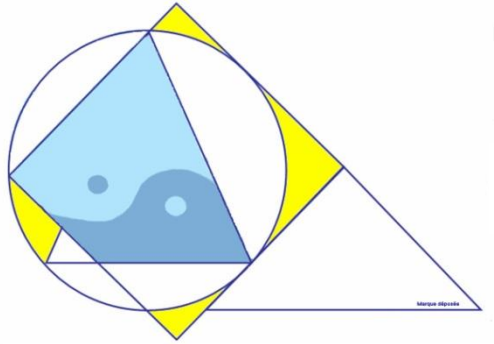
David Palermo

Mail : dpalermo1@myges.fr

Langage C

Semestre 1 : 24 heures

CAMPUS EDUCTIVE AIX-EN-PROVENCE



www.yantra-technologies.com

Le Langage C

Niveau 1


Facile


Normal


Difficile


Professionnel


Expert

david.palermo@yantra-technologies.com

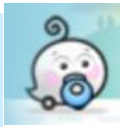
https://wiki.waze.com/wiki/Your_Rank_and_Points

Programmation en C : Sommaire

1 - Généralités

- Historique
- Les styles de programmation
- Avantages du langage C
- Inconvénients du langage C
- Environnement de développement

1 – Généralités -> Historique



- Le langage C a été créé au début des années 1970 par Dennis Ritchie (Laboratoires Bell AT&T). Son but était de ré-écrire le système d'exploitation Unix en langage evolué.
  En 1978, le premier standard est publié ("The C Programming Language") par B. W. Kernighan et D.M. Ritchie.

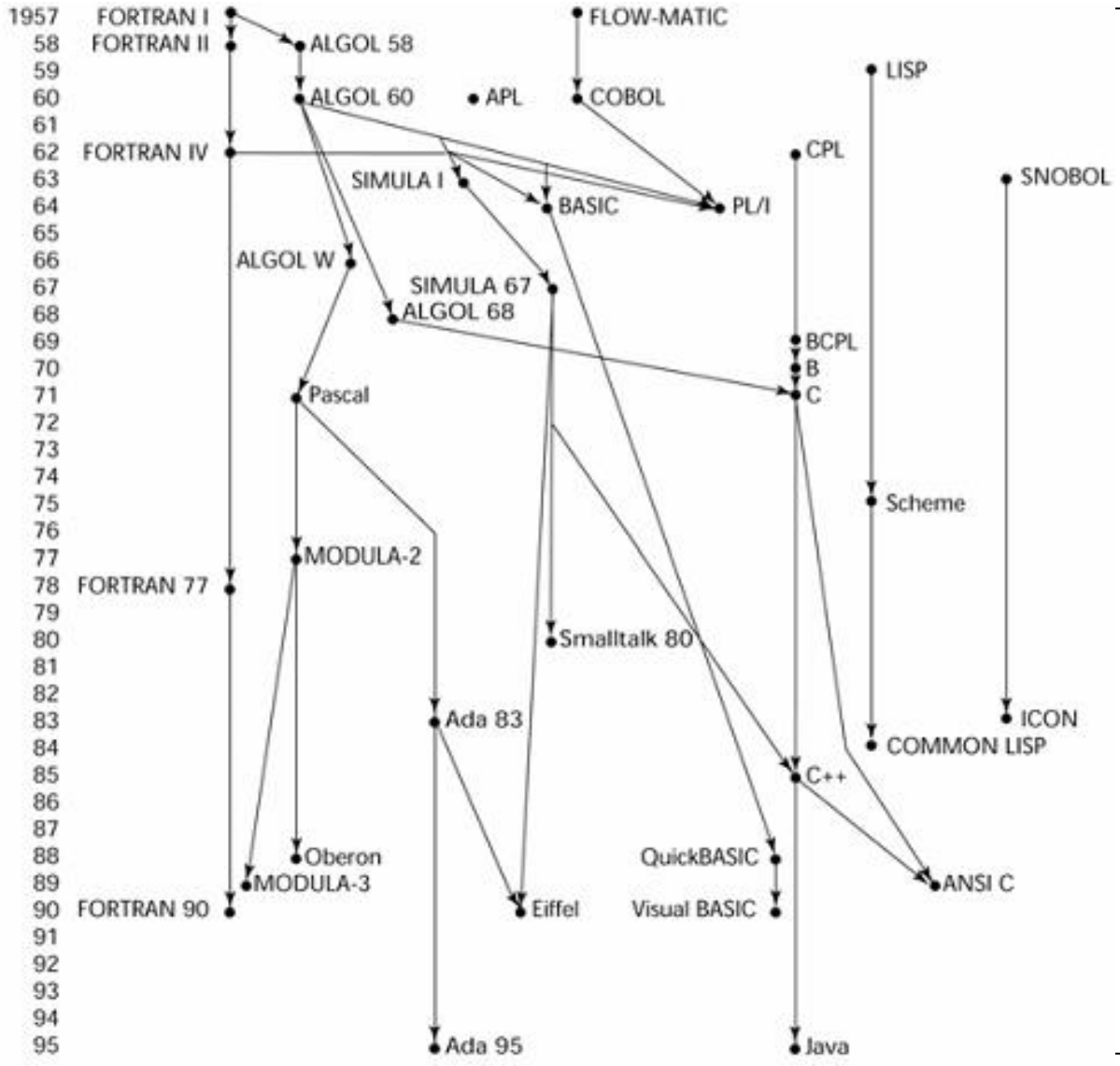
- En 1983, l‘ "American National Standards Institute" (ANSI) définit le standard ANSI-C qui est une définition explicite et indépendante de la machine pour le langage C".

- Le langage C++ est né en 1983 et a pour but de développer un langage qui garderait les avantages de ANSI-C (portabilité, efficacité) et qui permettrait en plus la programmation orientée objet. Premier standard ANSI-C++ en 1998 corrigée en 2003,

- Nouvelle version 2011 nouvelle norme du C99 & C11

.

1 – Généralités -> Historique



1 – Généralités -> Les styles de programmation

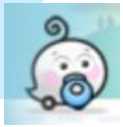
- **Fonctionnels : évaluation d 'expressions** : Lisp (1958), APL (récursivité) (1962), Caml (1985)

- **Procéduraux ou impératifs**: exécution d 'instructions

  Cobol (1959), *Fortran (1954), C (1970), Pascal (1970),*

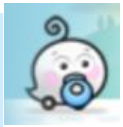
- **Objet** : *Simula (1964-1967) , Smalltalk (1976), C++ (1980), Java (1995), ADA 95 (1990), C# (2001)*
  - ensemble de composants autonomes (objets) qui disposent de moyens d 'interaction,
  - utilisation de classes,
  - échange de messages.

1 – Généralités -> Avantages du langage C



1. **Universel** : C n'est pas orienté vers un domaine d'applications spéciales
2. **Compact** : C est basé sur un noyau de fonctions et d'opérateurs limités qui permet la formulation d'expressions simples mais efficaces.
3. **Moderne** : C est un langage structuré, déclaratif et récursif. Il offre des structures de contrôle
4. **Près de la machine** : C offre des opérateurs qui sont très proches de ceux du langage machine (manipulations de bits, pointeurs…). C'est un atout essentiel pour la programmation des systèmes embarqués.
5. **Rapide** : C permet de développer des programmes concis et rapides.
6. **Indépendant de la machine** : C est un langage près de la machine (microprocesseur) mais il peut être utilisé sur n'importe quel système ayant un compilateur C. Au début C était surtout le langage des systèmes UNIX mais on le retrouve comme langage de développement aujourd'hui de tous les systèmes d'exploitation
7. **Portable** : En respectant le standard ANSI-C, il est possible d'utiliser (théoriquement ☺) le même programme sur tout autre système simplement en le recompilant. Il convient néanmoins de faire attention dans le cas d'un système embarqué qui n'inclut pas toujours un système d'exploitation…
8. **Extensible** : C peut être étendu et enrichi par l'utilisation de bibliothèques de fonctions achetées ou récupérées (logiciels libres…).

1 – Généralités -> Inconvénients du langage C



1. **Efficacité et compréhensibilité** : C autorise d'utiliser des expressions compactes et efficaces. Les programmes sources doivent rester compréhensibles pour nous-mêmes et pour les autres. ***Attention : La programmation efficace en langage C nécessite beaucoup d'expérience. Sans commentaires ou explications, les fichiers sources C peuvent devenir incompréhensibles et donc inutilisables (maintenance ?).***

2. **Portabilité et bibliothèques de fonctions :**
   - La portabilité est l'un des avantages les plus importants de C : en écrivant des programmes qui respectent le standard ANSI-C, nous pouvons les utiliser sur n'importe quelle machine possédant un compilateur ANSI-C.
   - Le nombre de fonctions standards ANSI-C (d'E/S) est limité. La portabilité est perdue si l'on utilise une fonction spécifique à la machine de développement.

3. **Discipline de programmation :**
   - C est un langage près de la machine donc dangereux bien que C soit un langage de programmation structurée.
   - C n'inclut pas de gestion automatique de la mémoire comme Java. ***La gestion mémoire en C est la cause de beaucoup de problèmes (***certains ont peut être eu une mauvaise expérience avec les *malloc()/free()*)…
   - L'instruction *goto* existe en C.

1 – Généralités -> Environnement de développement : compilateur

MinGW ou Mingw32 (https://www.mingw-w64.org/) (Minimalist GNU for Windows) est une adaptation des logiciels de développement et de compilation du GNU (GCC - GNU Compiler Collection), à la plate-forme Win32.

le compilateur de Visual Studio (https://visualstudio.microsoft.com/fr/)

GNU Compiler Collection, abrégé en GCC, (https://gcc.gnu.org/) est un ensemble de compilateurs créés par le projet GNU. GCC est un logiciel libre capable de compiler divers langages de programmation, dont C, C++, Objective-C, Java, Ada, Fortran et Go.

Clang (https://clang.llvm.org/) est un compilateur pour les langages de programmation C, C++ et Objective-C. (altenative au GCC)

Intel C++/C Compiler (https://software.intel.com) également connu sous les sigles icc et icl) désigne une gamme de compilateurs C++ développés par Intel, disponibles pour les plates-formes Linux, Microsoft Windows et Mac OS X.

1 – Généralités -> Environnement de développement : divers

Doxygen (https://www.doxygen.nl/index.html **) :** générateur de documentation sous licence libre capable de produire une documentation logicielle à partir du code source d'un programme.

Valgrind (https://www.valgrind.org/ **)** : est un outil de programmation libre pour déboguer, effectuer du profilage de code et mettre en évidence des fuites mémoires.

KCachegrind (http://kcachegrind.sourceforge.net/html/Home.html) est un outil d'affichage des données de profilage, utilisé pour déterminer les segments prenant le plus de temps lors de l'exécution d'un programme.

Intel VTune Performance Analyzer (https://software.intel.com **)** Profiler est un outil d'analyse des performances pour les machines x86 exécutant les systèmes d'exploitation Linux ou Microsoft Windows. D

Intel Parallel Studio (https://software.intel.com **)** est un logiciel propriétaire développé par Intel. Il permet de développer des logiciels en C++ et Fortran. S

TotalView Debugger (https://totalview.io/free-trial **)** est un débogueur pour applications écrites en C/C++ ou Fortran.

Voir : http://c.developpez.com/compilateurs/

1 – Généralités -> Environnement de développement : IDE

Eclipse CDT (C/C++ Development Tools) :

https://www.eclipse.org/cdt/

sous projet du projet "Eclipse Tools" dont le but est de fournir un environnement intégré de développement en C /C++ sous la forme de plug-ins pour Eclipse.

Code::Blocks (C/C++ et Fortran IDE)

https://www.codeblocks.org/

Environnement de développement intégré libre et multiplateforme C

Visual Studio Code

https://visualstudio.microsoft.com/fr/

IDE complet pour coder, déboguer, tester et déployer sur n'importe quelle plateforme.

https://www.codeur.com/blog/top-ide-environnement-de-developpement-c/
https://general.developpez.com/edi/
https://finobuzz.com/2017/07/04/les-10-ide-les-plus-connus-pour-programmer-en-c/

1 – Généralités -> Environnement de développement : IDE

CodeLite

https://codelite.org/

IDE intégré pour les langages C/C++, ¨PHP, node js . Il est libre, open source et multiplateforme.

Clion

https://www.jetbrains.com/fr-fr/clion/

Clion est un multiplateforme C / C ++ IDE pour Linux, OS X et Windows intégré avec le système de construction CMak

C++ Builder

https://www.embarcadero.com/fr/products/cbuilder

Borland C++ Compiler est une version gratuite du compilateur C & C++ inclus dans C++ Builder. Il est fourni avec les en-têtes Win32 ainsi que make.

https://www.codeur.com/blog/top-ide-environnement-de-developpement-c/
https://general.developpez.com/edi/
https://finobuzz.com/2017/07/04/les-10-ide-les-plus-connus-pour-programmer-en-c/

1 – Généralités -> Environnement de développement : IDE

NetBeans

https://netbeans.apache.org/kb/docs/cnd/

NetBeans est un IDE libre et gratuit, écrit en Java. Cet outil se compose d'une interface qui est munie d'une fonction de glisser-déposer et une liste de templates de projets préconstruits.

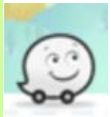
Anujta

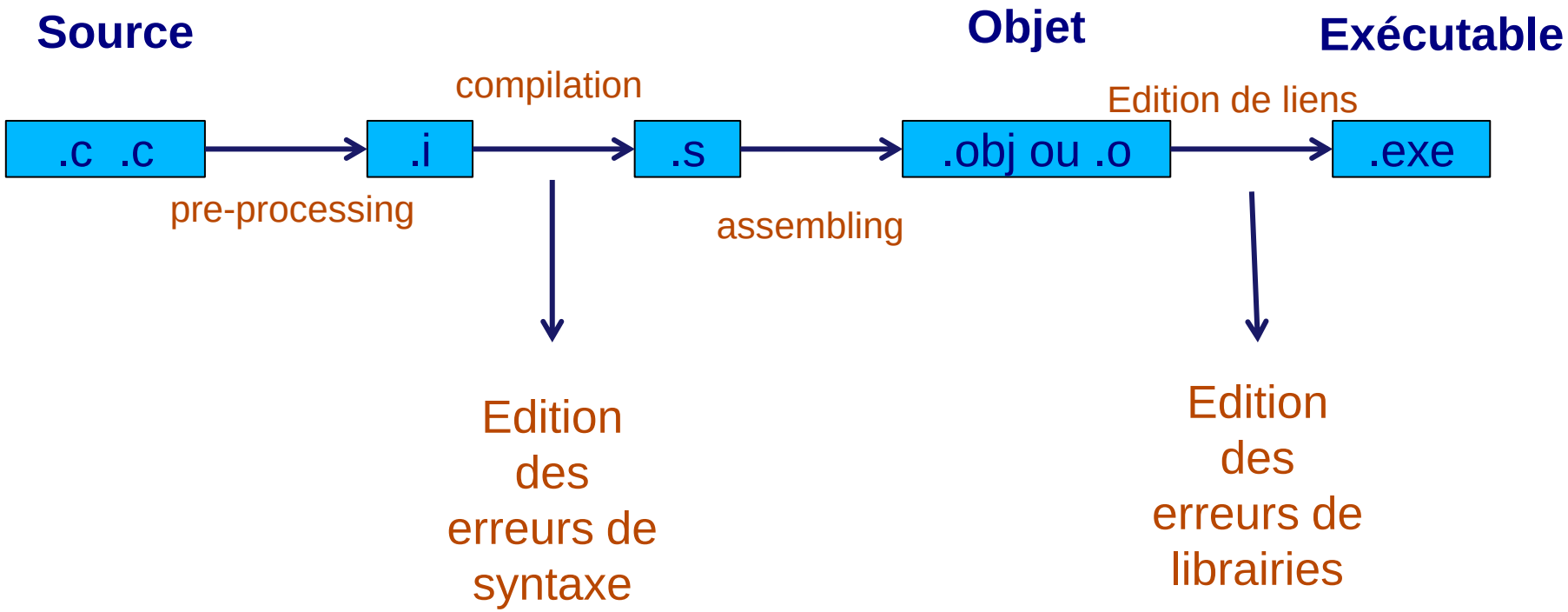
http://anjuta.org/

Anjuta est un IDE (les fonctionnalités RAD intégrant Glade sont en cour de développement) pour Linux développé dans le cadre du projet Gnome et par conséquent, il est particulièrement intéressant pour les développements en C avec GTK+ (coloration syntaxique des mots clés de GTK+) mais il peut aussi être utilisé pour le développement en C++ avec gtkmm

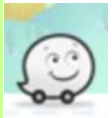
https://www.codeur.com/blog/top-ide-environnement-de-developpement-c/
https://general.developpez.com/edi/
https://finobuzz.com/2017/07/04/les-10-ide-les-plus-connus-pour-programmer-en-c/

2 – Compilation -> Etapes de compilation





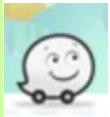
2 – Compilation -> Préprocesseur



- suppression des commentaires (// et /* */)
- inclusion des fichiers .h ou .hh dans le fichier .c ou .c …. (#include)
- traitement des directives de compilation qui commencent par un caractère #.

gcc -E bonjour.c > bonjour.i ou gcc -E bonjour.c > bonjour.i

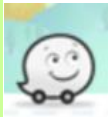
2 – Compilation -> Compilation



La Compilation le code ascii en code assembleur.

gcc -S bonjour.i ou gcc -S bonjour.i création d'un bonjour.o

2 – Compilation -> Assembling

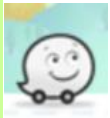


La Compilation le code ascii en code assembleur.

gcc bonjour.o ou gcc bonjour.o création d'un exécutable a.out

gcc –o bonjour bonjour.o ou gcc -o bonjour bonjour.o création d'un exécutable bonjour

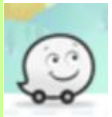
2 – Compilation -> Edition de lien



La Compilation le code ascii en code assembleur.

gcc –c bonjour.s ou gcc -c bonjour.s création de bonjour.o

2 – Compilation -> Compilation simplifier



gcc –c bonjour.c

gcc –o bonjour bonjour.o création de l'exécutable bonjour

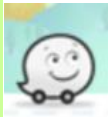
gcc –o bonjour bonjour.c   création de l'exécutable bonjour

-c : Compile les fichiers sources, mais sans édition de liens. Le compilateur produit un fichier *objet* .o pour chaque fichier source.

-o *fichRes* : Donne le nom *fichRes* au fichier exécutable produit par la compilation et l'édition de liens. Quand cette option n'est pas employée, le nom d'exécutable par défaut est a.out.

-g : engendre de l'information pour le débogage, à utiliser avec un débogueur comme gdb (ou son interface utilisateur ddd).

2 – Compilation -> Construction de bibliothèques



Bibliothèques statiques

En Unix ou Linux, pour réunir des fichiers objets, on utilisera typiquement la commande ar et ranlib, qui permet de créer des fichiers «archives» :

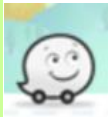
```
gcc –o F_1.o –c F_1.c
ar  -rv libmabib.a F_1.o F_2.o F_3.o
ranlib libmabib.a
```

Bibliothèques dynamiques

En Unix ou Linux, pour réunir des fichiers objets, on utilisera l'option spécial –fPIC pour compiler les objets et –shared à l'édition de lien:

```
gcc –fPIC –o F_1.o –c F_1.c
gcc -shared -fPIC -o libmabib.so F_1.o F_2.o F_3.o
```

Utilisation de bibliothèques



Bibliothèques statiques

gcc –o mon_excutable main.c /chemin/libmabib.a

ou

gcc –static –o mon_excutable main.c –L/chemin **–l**mabib

Bibliothèques dynamiques

gcc –o mon_excutable main.c /chemin/libmabib.so

ou

gcc –o mon_excutable main.c –L/chemin **–l**mabib

Ne pas oublier de mettre à jour la variable LD_LIBRARY_PATH
LD_LIBRARY_PATH=$ LD_LIBRARY_PATH:/chemin

2 – Compilation -> Mon premier programme

```
#ifndef Bonjour_H
#define Bonjour_H

void Bonjour()  ;

#endif
```

Bonjour.h

```
#include "Bonjour.h"
#include <stdio.h>

void Bonjour()  {
      printf("Bonjour ! \n");
}
```

Bonjour.c

2 – Compilation -> Mon premier programme

```
#include "Bonjour.h"

/* Mon premier programme  */

int main (int argc, char *argv[])
{
    Bonjour();
     return 0;
}
```

main.c

2 – Compilation -> Ma première compilation

gcc -c Bonjour.c

=> crée un fichier objet Bonjour.o

gcc -o Bonjour.exe main.c Bonjour.o

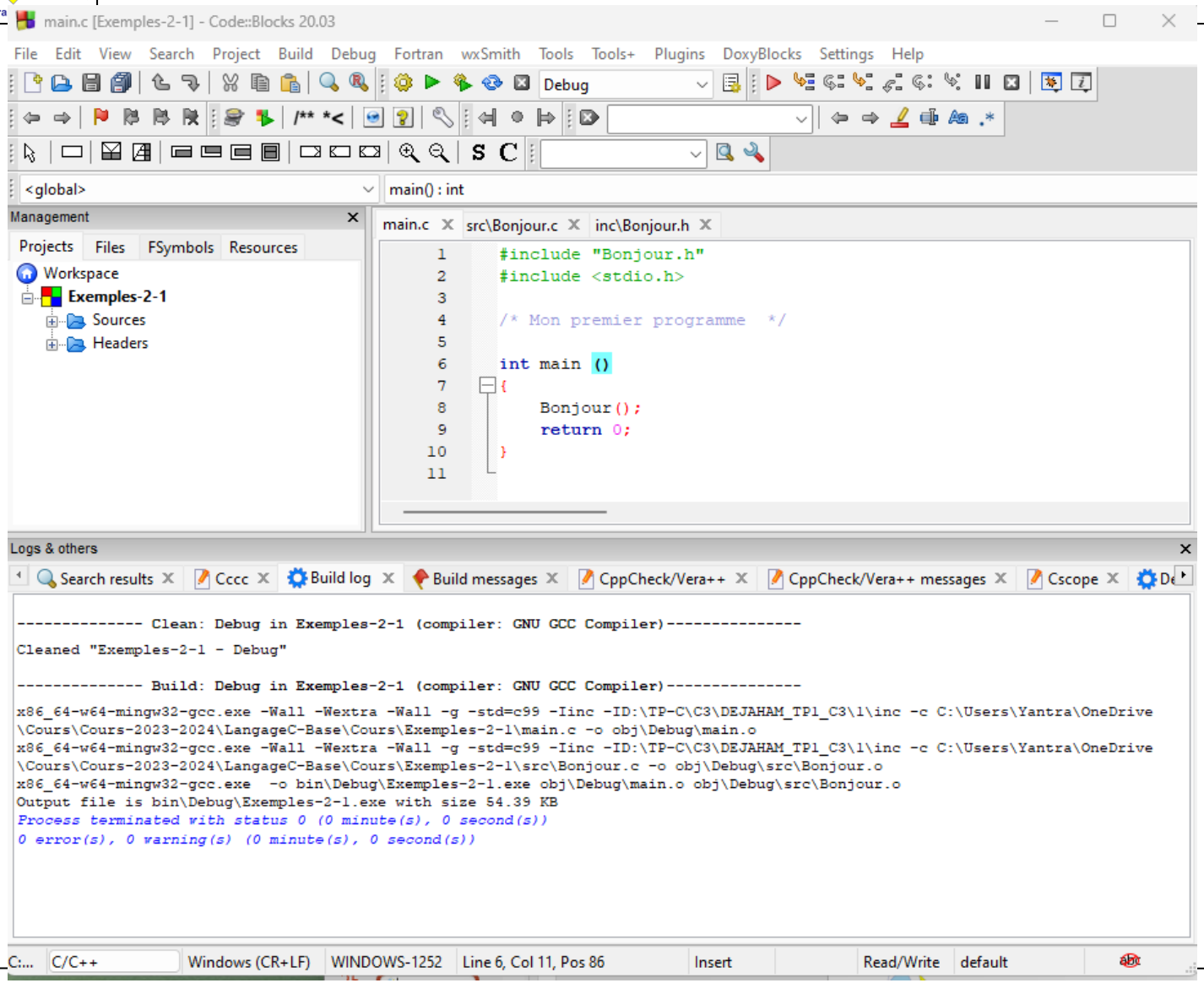
=> crée un exécutable Bonjour.exe

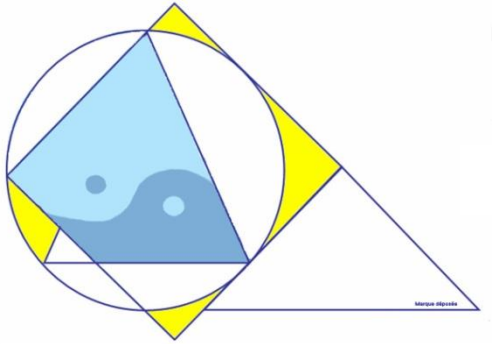
https://www.mingw-w64.org/

https://korben.info/comment-installer-gcc-sous-osx-sans-installer-xcode.html

https://apps.apple.com/fr/app/xcode/id497799835?mt=12

2 – Compilation -> Code::Blocks





www.yantra-technologies.com

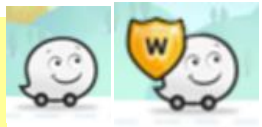
3 – Le Langage C

david.palermo@yantra-technologies.com

Références:

https://www.ltam.lu/Tutoriel-Ansi-C/
https://www.cplusplus.com/
http://public.iutenligne.net/informatique/algorithme-et-programmation/priou/LangageC/index.html

3 – Le Langage C-> Sommaire



- Les mots clé du langage
- Les commentaires
- Les types prédéfinis
- Les variables
- Les opérateurs
- Les structures de contrôle
- Les tableaux static
- les chaînes de caractères static
- Les fonctions
- Les pointeurs
- Les types de variables complexes

3 – Le Langage C -> Mots-clés du C++/C

alignas (depuis C++11)	char32_t (depuis C++11)	enum	namespace	return	try
alignof (depuis C++11)	class	explicit	new	short	typedef
and	compl	export	noexcept (depuis C++11)	signed	typeid
and_eq	const	extern	not	sizeof	typename
asm	constexpr (depuis C++11)	false	not_eq	static	union
auto (depuis C++11)	const_cast	float	nullptr (depuis C++11)	static_assert (depuis C++11)	unsigned
bitand	continue	for	operator	static_cast	using
bitor	decltype (depuis C++11)	friend	or	struct	virtual
bool	default	goto	or_eq	switch	void
break	delete	if	private	template	volatile
case	do	inline	protected	this	wchar_t
catch	double	int	public	thread_local (depuis C++11)	while
char	dynamic_cast	long	register	throw	xor
char16_t (depuis C++11)	else	mutable	reinterpret_cast	true	xor_eq
override (C++11)	final (C++11)				

3 – Le Langage C-> Les commentaires

Commentaire C :

```
/* Ceci est un commentaire C */
```

Depuis C99

```
// ligne
```

https://www.doxygen.nl/
https://franckh.developpez.com/tutoriels/outils/doxygen/

3 – Le Langage C -> Types prédéfinis

Type	Description	Taille (*)	Valeur
void	Type générique		
char	caractère/octet (†)	1 octets	comme signed ou unsigned char
unsigned char	caractère/octet (†) non signé	1 octets	0 **à** 255
signed char	caractère/octet (†) signé	1 octets	-128 **à** 127[1][2]
short	entier court signé	2 octets	-32 768 **à** 32 767
unsigned short	entier non signé	2 octets	0 **à** 65 535
int	entier signé	2 ou 4 octets	(en fonction du compilateur)
unsigned	entier non signé	2 ou 4 octets	(en fonction du compilateur)
long	entier signé long	4 octets	-2 147 483 648 **à** 2 147 483 647
unsigned long	entier non signé long	4 octets	0 **à** 4 294 967 295 (2^32-1)
float	flottant (‡)	4 octets	Mantisse : +- 6 chiffres significatifs
double	flottant (‡)	8 octets	Mantisse : +- 12 chiffres significatifs
long double	flottant (‡)	10 octets	

- les tableaux à une dimension, dont les indices sont spécifiés par des crochets (’[’ et ’]’).
- les structures, unions et énumérations, les fonctions

Remarque :

- la taille des types peut dépendre des compilateurs,
- en hexadécimal, ils s’écrivent de la manière suivante : 127 -> 0x7f
- un bit est réservé pour le signe

https://fr.wikiversity.org/wiki/Introduction_au_langage_C/Types

3 – Le Langage C
-> Types prédéfinis : <limits.h> & <float.h>

La norme nous fournit deux en-têtes définissant des macros constantes correspondant aux limites des types entiers et flottants : <limits.h> et <float.h>.

https://cplusplus.com/reference/climits/
https://cplusplus.com/reference/cfloat/

3 – Le Langage C
-> Types prédéfinis : <limits.h> & <float.h>

name	expresses	possible value*
CHAR_BIT	Number of bits in a char object (byte)	8 or greater*
SCHAR_MIN	Minimum value for an object of type signed char	-127 (-2^7+1) or less*
SCHAR_MAX	Maximum value for an object of type signed char	127 (2^7-1) or greater*
UCHAR_MAX	Maximum value for an object of type unsigned char	255 (2^8-1) or greater*
CHAR_MIN	Minimum value for an object of type char	either SCHAR_MIN or 0
CHAR_MAX	Maximum value for an object of type char	either SCHAR_MAX or UCHAR_MAX
MB_LEN_MAX	Maximum number of bytes in a multibyte character, for any locale	1 or greater*
SHRT_MIN	Minimum value for an object of type short int	-32767 ($-2^{15}+1$) or less*
SHRT_MAX	Maximum value for an object of type short int	32767 ($2^{15}-1$) or greater*
USHRT_MAX	Maximum value for an object of type unsigned short int	65535 ($2^{16}-1$) or greater*
INT_MIN	Minimum value for an object of type int	-32767 ($-2^{15}+1$) or less*
INT_MAX	Maximum value for an object of type int	32767 ($2^{15}-1$) or greater*
UINT_MAX	Maximum value for an object of type unsigned int	65535 ($2^{16}-1$) or greater*
LONG_MIN	Minimum value for an object of type long int	-2147483647 ($-2^{31}+1$) or less*
LONG_MAX	Maximum value for an object of type long int	2147483647 ($2^{31}-1$) or greater*
ULONG_MAX	Maximum value for an object of type unsigned long int	4294967295 ($2^{32}-1$) or greater*
LLONG_MIN	Minimum value for an object of type long long int	-9223372036854775807 ($-2^{63}+1$) or less*
LLONG_MAX	Maximum value for an object of type long long int	9223372036854775807 ($2^{63}-1$) or greater*
ULLONG_MAX	Maximum value for an object of type unsigned long long int	18446744073709551615 ($2^{64}-1$) or greater*

3 – Le Langage C

> Types prédéfinis : <limits.h> & <float.h>

name	value	stands for	expresses
FLT_RADIX	2 or greater	RADIX	Base for all floating-point types (float, double and long double).
FLT_MANT_DIG DBL_MANT_DIG LDBL_MANT_DIG		MANTissa DIGits	Precision of *significand*, i.e. the number of digits that conform the *significand*.
FLT_DIG DBL_DIG LDBL_DIG	6 or greater 10 or greater 10 or greater	DIGits	Number of **decimal digits** that can be rounded into a floating-point and back without change in the number of decimal digits.
FLT_MIN_EXP DBL_MIN_EXP LDBL_MIN_EXP		MINimum EXPonent	Minimum negative integer value for the *exponent* that generates a normalized floating-point number.
FLT_MIN_10_EXP DBL_MIN_10_EXP LDBL_MIN_10_EXP	-37 or smaller -37 or smaller -37 or smaller	MINimum base-10 EXPonent	Minimum negative integer value for the *exponent* of a base-10 expression that would generate a normalized floating-point number.
FLT_MAX_EXP DBL_MAX_EXP LDBL_MAX_EXP		MAXimum EXPonent	Maximum integer value for the *exponent* that generates a normalized floating-point number.
FLT_MAX_10_EXP DBL_MAX_10_EXP LDBL_MAX_10_EXP	37 or greater 37 or greater 37 or greater	MAXimum base-10 EXPonent	Maximum integer value for the *exponent* of a base-10 expression that would generate a normalized floating-point number.
FLT_MAX DBL_MAX LDBL_MAX	1E+37 or greater 1E+37 or greater 1E+37 or greater	MAXimum	Maximum finite representable floating-point number.
FLT_EPSILON DBL_EPSILON LDBL_EPSILON	1E-5 or smaller 1E-9 or smaller 1E-9 or smaller	EPSILON	Difference between 1 and the least value greater than 1 that is representable.
FLT_MIN DBL_MIN LDBL_MIN	1E-37 or smaller 1E-37 or smaller 1E-37 or smaller	MINimum	Minimum representable positive floating-point number.
FLT_ROUNDS		ROUND	Rounding behavior. Possible values: -1 undetermined 0 toward zero 1 to nearest 2 toward positive infinity 3 toward negative infinity Applies to all floating-point types (float, double and long double).
FLT_EVAL_METHOD		EVALuation METHOD	Properties of the evaluation format. Possible values: -1 undetermined 0 evaluate just to the range and precision of the type 1 evaluate float and double as double, and long double as long double. 2 evaluate all as long double Other negative values indicate an implementation-defined behavior. Applies to all floating-point types (float, double and long double).
DECIMAL_DIG		DECIMAL DIGits	Number of decimal digits that can be rounded into a floating-point type and back again to the same decimal digits, without loss in precision.

3 – Le Langage C
-> Types prédéfinis : Quelques constantes caractères

le type char est un type entier particulier

caractère	signification	caractère	signification	caractère	signification	caractère	signification
\a	Sonnerie	\\	Trait oblique	\b	Curseur arrière	\?	Point d'interrogation
\t	Tabulation	\'	Apostrophe	\n	Nouvelle ligne	\"	guillemets
\r	retour au début de ligne	\f	saut de page (imprimante)	\0	Fin d'une chaine de caractère	\v	Tabulation verticale

https://fr.vikidia.org/wiki/Codage_des_caract%C3%A8res
https://www.france-ioi.org/algo/course.php?idChapter=590&idCourse=2299

3 – Le Langage C
-> Types prédéfinis : Déclaration type réel

Forme :

<+/-> mantisse * [E,e] <exposant>

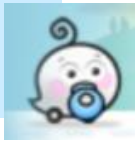
Composés :
- d'un signe : + ou -
- d'une mantisse : chiffre positif à 1 chiffre avant la virgule
- d'un exposant : entier relatif

3 principaux types : float, double, long float

Exemples :

- double d = -1.E5;

3 – Le Langage C
-> Les variables : Déclarer une variable



- Forme
  - type nomdevariable**;**
- Exemples
  - int i; /* déclaration de la variable i */
  - char c; /* déclaration du caractère c */

3 – Le Langage C
-> Les variables : Initialiser une variable

C permet l'initialisation des variables dans la zone de déclaration

Affecter		Initialiser
char c; c='A';	abouti au même résultat que	char c='A';
int i; i=50;	abouti au même résultat que	int i=50;
const int i;	n'abouti pas au même résultat que	**const int i = 50;**

i = 50; // erreur

3 – Le Langage C -> Les variables : const

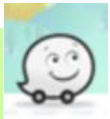
Qualificatif **const**

Une variable dont le type est qualifiée par **const** ne peut pas être modifiée.

But : se protéger contre une erreur de programmation.

Ceci est utile pour les paramètres d'une fonction, lorsqu'on désire protéger les paramètres effectifs de la fonction en les mettant en **lecture seulement** pour la fonction.

3 – Le Langage C -> Les opérateurs



- Les opérateurs : arithmétiques
- Les opérateurs : simplifiés
- Les opérateurs : logiques
- Les opérateurs : conditionnels
- Les opérateurs : binaires
- Opérateur sizeof()
- Opérateur de cast ()

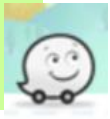
3 – Le Langage C -> Les opérateurs : arithmétiques

- Sur les réels : + - / *
- Sur les entiers : + - / (quotient de la division), % (reste de la division)

l'affectation se fait grâce au signe '='

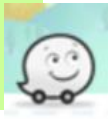
Hiérarchie habituelle (* et / prioritaires par rapport aux + et -)

3 – Le Langage C -> Les opérateurs : simplifiés



- i=i+1 peut s'écrire i++; (++i;)
- i=i-1 peut s'écrire i--; (--i;)
- a=a+b peut s'écrire a+=b;
- a=a-b peut s'écrire a-=b;
- a=a&b peut s'écrire a&=b;

3 – Le Langage C -> Les opérateurs : logiques

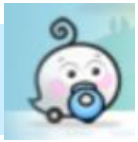


- Quelle utilisation ?
- Opérateurs classiques
- Les expressions
- Et - Ou - !

3 – Le Langage C -> Les opérateurs : logiques -> Quelle utilisation ?

- Tous les tests :
  - si (expression vraie) ...
  - tant que (expression vraie) ...

- Valeurs booléennes
  - Vrai (1)
  - Faux (0)



- Egalité

  a == b

- Inégalité

  a != b

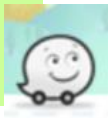
- Relations d'ordre

  a < b a <= b a > b a >= b

- Expression

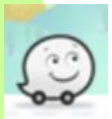
  a est vrai si a est différent de 0

3 – Le Langage C -> Les opérateurs : logiques -> Et - Ou - !



- Et Logique
  - expression1 && expression2
  - si expression1 et expression2 sont vraies
- Ou Logique
  - expression1 || expression2
  - si expression1 ou expression2 est vraie
- ! (Non)
  - !expression
  - si expression est fausse

3 – Le Langage C -> Les opérateurs : logiques -> Et - Ou - !



- Le 'ET'

A	B	ET
0	0	0
0	1	0
1	0	0
1	1	1

- Le 'OU'

A	B	OU
0	0	0
0	1	1
1	0	1
1	1	1

3 – Le Langage C -> Les opérateurs : conditionnels

expr1 ? expr2 : expr3

si (expr1) alors expr2 sinon expr3

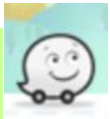
Exemple : fonction min

```
if (y<z)  x=y; else x=z;
```

Equivalent à

```
x = ( y<z) ? y : z ;
```

3 – Le Langage C -> Les opérateurs : binaires



- &, et (and)

Table de vérité AND

A	B	A AND B
0	0	0
0	1	0
1	0	0
1	1	1

- |, ou, (OR)

Table de vérité OR

A	B	A OR B
0	0	0
0	1	1
1	0	1
1	1	1

- ^, ou exclusif (XOR)

Table de vérité XOR

A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

3 – Le Langage C -> Les opérateurs : binaires

Table de vérité NOT

A	NOT A
0	1
1	0

- ~, complément à 1

- <<, décalage à gauche
- >>, décalage à droite

Exemple : p = n « 3;

p égal n décalé de 3 bits sur la gauche

https://zestedesavoir.com/tutoriels/755/le-langage-c-1/notions-avancees/manipulation-des-bits/

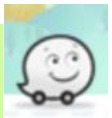
3 – Le Langage C -> Les opérateurs : binaires

```c
int main ()
{
    int a = 0x63; /* 0x63 == 99 == 0110 0011 */
    int b = 0x2A; /* 0x2A == 42 == 0010 1010 */

    printf("a == %2X == %d \n",a,a );
    printf("b == %2X == %d \n",b,b );

    /* 0110 0011 & 0010 1010 == 0010 0010 == 0x22 == 34 */
    printf("a & b == %2X == %d \n", a & b, a & b);
    /* 0110 0011 | 0010 1010 == 0110 1011 == 0x6B == 107 */
    printf("a | b == %2X == %d \n",a | b, a | b);
    /* 0110 0011 ^ 0010 1010 == 0100 1001 == 0x49 == 73 */
    printf("a ^ b == %2X == %d \n",a ^ b, a ^ b);
    /* ~0110 0011 == 1001 0011 */
    printf("~a == %2X == %d \n",~a,~a);
    /* 0110 0011 << 3 == 01 1000 11000 */
    printf("a<<3 == %2X == %d \n",a<<3,a<<3);
    /* 0110 0011 >> 2 == 0001 1000 */
    printf("a>>2 == %2X == %d \n",a>>2,a>>2);
    return 0;
}
```

```
a == 63 == 99
b == 2A == 42
a & b == 22 == 34
a | b == 6B == 107
a ^ b == 49 == 73
~a == FFFFFF9C == -100
a<<3 == 318 == 792
a>>2 == 18 == 24
```

https://zestedesavoir.com/tutoriels/755/le-langage-c-1/notions-avancees/manipulation-des-bits/
https://sebastienguillon.com/test/javascript/convertisseur.html
http://pedagogie.ac-limoges.fr/sti_si/accueil/FichesConnaissances/Sequence2SSi/co/ConvDecimalBinaire.html

3 – Le Langage C -> Les opérateurs : sizeof()



std::size_t sizeof(type)

Operateur qui rends la taille de l'objet ou de type passé en paramètre

```
#include <iostream>
struct Empty {};
struct Bit {unsigned bit:1; };
int main()
{
    Empty e;
    Bit b;
    std::cout << "size of empty class: "     << sizeof e       << '\n'
              << "size of pointer : "        << sizeof &e      << '\n'
//            << "size of function: "        << sizeof(void())  << '\n'  // compile error
//            << "size of incomplete type: " << sizeof(int[])   << '\n'  // compile error
//            << "size of bit field: "       << sizeof b.bit    << '\n'  // compile error
              << "size of array of 10 int: " << sizeof(int[10]) << '\n';
}
```

Résultat :

```
size of empty class: 1
size of pointer : 8
size of array of 10 int: 40
```

https://fr.cppreference.com/w/cpp/language/sizeof

3 – Le Langage C -> Les opérateurs : Opérateur de cast ()

L'opérateur de conversion de type s'appelle un cast.

Un cast de type fournit une méthode pour la conversion explicite du type d'un objet dans u

```
#include <iostream>

using namespace std;

int main()
{
    double x = 3.1;
    int i;
    cout << "x = " << x << endl;
    i = (int)x;   // assign i the integer part of x
    cout << "i = " << i << endl;
}
```

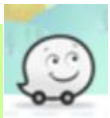
3 – Le Langage C -> Les opérateurs : Priorité des opérateurs

priorité	Opérateur	Associativité
16	() [] -> . ++ [1] -- [2]	G
15	! ~ ++ [3] -- [4] - [5] + [6] * [7] & [8] sizeof	D
14	*conversion*	D
13	* [9] / %	G
12	+ -	G
11	<< >>	G
10	< <= > >=	G
9	== !=	G
8	& [10]	G
7	^	G
6	|	G
5	&&	G
4	||	G
3	?:	D
2	= += -= *= /= %= >>= <<= &= ^= |=	D
1	,	G

```c
int main ()
{
    int i=3;
    printf(" 1 + 2 * 5 + ++i / i = %d \n",(1 + 2 * 5 + ++i / i ));
    printf(" 1 + 2 * 5 + i++ / i = %d \n",(1 + 2 * 5 + i++ / i ));
    return 0;
}
```

https://c.developpez.com/cours/bernard-cassagne/node101.php

3 – Le Langage C -> Les structures de contrôle



- Si … alors … sinon … => if () {} else {}
- Au cas … ou faire … => switch () { case }
- Tant que … faire … => while () {}
- Répéter … tant que …=> do {} while ()
- Boucle Pour … faire … => for () {}
- Saut => goto
- Rupture de séquence => break, continue, return

3 – Le Langage C -> Les structures de contrôle : si … alors … sinon … => if () {} else {}

Si (expression conditionnelle vraie)
alors { instructions 1 }
sinon { instructions 2 }

```
if (expression) {
   instructions 1;
}
else {
   instructions 2;
}

if (expression) {
   instructions 1;     /* else optionnel */
}
```

```
if (expression)
  instruction;     /* Une instruction */

if (expression) instruction1;
else instruction2;
```

https://zestedesavoir.com/tutoriels/755/le-langage-c-1/1042_les-bases-du-langage-c/4294_les-selections/

3 – Le Langage C -> Les structures de contrôle : si … alors … sinon … => if () {} else {} - Exemple

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

int main(int argc, char *argv[]) {
    char quit = '\0';
    printf(" Bonjour ! \n");
    printf(" Appuyer sur la touche q pour quitter \n");
    quit=getchar();
    if ( quit == 'q' ) return 0;

    if ( quit == 'Q' ) {
     printf(" Appuyer sur la touche q et pas Q pour quitter \n");
     quit=getchar();
    }
    else
    {
     printf("  Appuyer sur la touche q pour quitter \n");
     quit=getchar();
    }

    if ( quit == 'q' ) return 0;
    else printf("Dernier essai, Appuyer sur la touche q pour quitter \n") ;

    quit=getchar();
    return 0;
}
```

3 – Le Langage C -> Les structures de contrôle : Au cas … ou faire …=> switch () { case }

Au cas ou la variable vaut :

valeur 1 : faire … ;

valeur 2 : faire … ; etc.

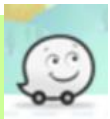
```
switch (variable de type char ou int)  {
    case valeur 1 : …;
            …;
            break;
    case valeur 2 : …;
            …;
            break;
    default :          …;
            …;
            break;
}
```

https://zestedesavoir.com/tutoriels/755/le-langage-c-1/1042_les-bases-du-langage-c/4294_les-selections/

3 – Le Langage C -> Les structures de contrôle : Au cas … ou faire …=> switch () { case } - Exemple

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

int main(int argc, char *argv[])
{
   char quit = '\0';
   printf ( " Bonjour ! \nAppuyer sur la touche q pour quitter \n");
   quit=getchar();
   switch ( quit )
   {
      case 'q' :
            return 0; break;
      case 'Q' :
            printf ( " Appuyer sur la touche q et pas %c pour quitter \n",quit);
          quit=getchar();
            break;
      case ‘r’ :
      case ‘R’ :  case 'S' : case 'T' :
            printf ( " Appuyer sur la touche q et pas %c pour quitter \n",quit);
          quit=getchar();
            break;
      default :
            printf ( " Appuyer sur la touche q et pas %c pour quitter \n",quit);
            quit=getchar();
            break;
   }

   if ( quit == 'q' ) return 0;
   else printf ( "Dernier essai, Appuyer sur la touche q pour quitter \n");
   quit=getchar();
   return 0;
}
```

3 – Le Langage C -> Les structures de contrôle : Tant que … faire => while () {}

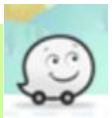


Tant que (expression vraie) faire
{bloc d 'instructions}

```
while (expression vraie)  {
     bloc d 'instructions }
```

https://zestedesavoir.com/tutoriels/755/le-langage-c-1/1042_les-bases-du-langage-c/4295_les-boucles/#1-12885_la-boucle-while

3 – Le Langage C -> Les structures de contrôle : Tant que … faire => while () {} - Exemple



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

int main(int argc, char *argv[])
{
    char quit = '\0';
    while ( quit != 'q' )
    {
      printf ( " Bonjour ! \n Appuyer sur la touche q pour quitter \n");
      quit=getchar();
    }
    return 0;
}
```

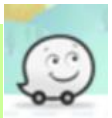
3 – Le Langage C -> Les structures de contrôle : Répéter … tant que …=> do {} while ()

Répéter {bloc d 'instructions}
tant que (expression vraie)

```
do {bloc d 'instructions} while
(expression vraie);
```

https://zestedesavoir.com/tutoriels/755/le-langage-c-1/1042_les-bases-du-langage-c/4295_les-boucles/#2-12886_la-boucle-do-while

3 – Le Langage C -> Les structures de contrôle : Répéter … tant que …=> do {} while () - Exemple



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

int main(int argc, char *argv[])
{
    char quit = '\0';
    do
    {
      printf ( " Bonjour ! \n Appuyer sur la touche q pour quitter \n");
      quit=getchar();
    }
    while ( quit != 'q' );

    return 0;
}
```

3 – Le Langage C -> Les structures de contrôle : Boucle Pour … faire … => for () {}

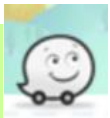
Pour (initialisation;condition;modification) faire {

bloc d 'instructions}

```
for (initialisation;condition;modification)  {
    bloc d 'instructions}
```

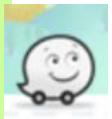
https://zestedesavoir.com/tutoriels/755/le-langage-c-1/1042_les-bases-du-langage-c/4295_les-boucles/#3-12887_la-boucle-for

3 – Le Langage C -> Les structures de contrôle : Boucle Pour … faire … => for () {} - Exemple



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

int main(int argc, char *argv[])
{
    char quit = '\0';
    int i;
    printf ( " Bonjour ! \n Appuyer sur la touche q pour quitter \n");
    quit=getchar();
    for ( ; quit!='q'; )
    {
        for (i = 0 ; i < 5; i++) printf ( "Bonjour ! \n");
        printf ( " Appuyer sur la touche q pour quitter => ");
        scanf("%c",&quit);
        getchar();
     }
    return 0;
}
```

3 – Le Langage C -> Les structures de contrôle : Saut => goto

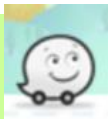


```
goto etiquette ;
etiquette :
```

https://zestedesavoir.com/tutoriels/755/le-langage-c-1/1042_les-bases-du-langage-c/4610_les-sauts/#4-13494_linstruction-goto

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

int main(int argc, char *argv[])
{
    char quit = '\0';

    reprise :

    printf ( " Bonjour ! \n Appuyer sur la touche q pour quitter \n");
      scanf("%c",&quit);getchar();
      if ( quit!='q' ) goto reprise;

    return 0;
}
```

3 – Le Langage C -> Les structures de contrôle : Rupture de séquence => break, continue, return

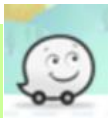


- ***Return[valeur]*** : permet de quitter immédiatement la fonction en cours.

- ***break*** : permet de passer à l'instruction suivant l'instruction *while*, *do*, *for* ou *switch* la plus imbriquée.

- ***continue*** : saute directement à la dernière ligne de l'instruction *while*, *do* ou *for* la plus imbriquée.

3 – Le Langage C -> Les structures de contrôle : Rupture de séquence => break, continue, return – Exemple



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

int main(int argc, char *argv[])
{
   char quit = '\0';
   int i;

   while ( quit!='q' )
   {
      for (i=0;i< 5;i++)
      {
            printf ( " Bonjour ! : num => %i \n",i);
            printf ( " Appuyer sur la touche 'o' pour quitter la boucle\n");
            scanf("%c",&quit);getchar();
            if ( quit=='o' ) break;
      }
   printf ( " Appuyer sur la touche 'r' pour recommencer la boucle\n");
   printf ( " Appuyer sur la touche 'q' pour quitter \n");
   scanf("%c",&quit);getchar();

   if ( quit=='r' ) continue;
   if ( quit!='q' ) return 0;
   if ( quit== 's') break;

   }
   return 0;
}
```

3 – Le Langage C -> Les tableaux statique : Définition

Tableau caractérisé par

- sa taille
- ses éléments repérés par son indice
- commence à 0 finit à dim-1

Tableau à une dimension

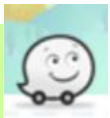
- Déclaration : Type nom[dim]; /* réservation de dim places de grandeur Type */
- Exemples
  - int Tab[10];
  - float vecteur[20];

Tableau à deux dimensions

- Déclaration : type nom[dim1][dim2];
  - int compteur[4][5];
  - float nombre[2][10];

- Utilisation : nom[indice1][indice2];
  - compteur[3][4]=3;
  - printf("%d",compteur[1][2]);

3 – Le Langage C -> Les tableaux statique : Déclaration



On peut initialiser au moment de la déclaration

- Exemple
  - **int liste[10]={1,2,45,5,67,120,32,48,3,10};**
  - **int tab[2][3]={{1,4,8},{8,5,3}}**

Remarque

- Consomment beaucoup de place
- Nécessité de les dimensionner au plus juste

3 – Le Langage C -> Les tableaux statique : Exemple

```c
int main ()
{

    // Déclaration d'un tableau statique de taille 5 pour stocker des nombres entiers
    int numbers[5] = {10, 20, 30, 40, 50};

    // Accès aux éléments du tableau
    printf("Element 1: %d\n", numbers[0]);
    printf("Element 3: %d\n", numbers[2]);

    // Modification d'un élément du tableau
    numbers[1] = 25;
    printf("Modified Element 2: %d\n", numbers[1]);

    // Parcours et affichage des éléments du tableau
    printf("Array elements: ");
    for (short unsigned i = 0; i < 5; ++i) {
        printf("%d ", numbers[i]);
    }
    printf("\n");


}
```

```
Element 1: 10
Element 3: 30
Modified Element 2: 25
Array elements: 10 25 30 40 50
```

3 – Le Langage C -> Les chaînes de caractères static

- Une chaîne = un tableau de caractères
- Déclaration
  - **char nom[dim];**
  - **exemple : char texte[10]**
- Seulement **dim-1** caractères disponibles
- Caractère NUL (‘ \0 ’) à la fin

3 – Le Langage C -> Les chaînes de caractères static : Affichage et saisie d ’une chaîne

- printf et format %s
  - Exemple : **char texte[10]=“bonjour“;**
    printf(“voici le texte %s“,texte);
- scanf et format %s
  - Exemple : **char texte[10];**
    scanf(“%s“,texte);

3 – Le Langage C -> Les chaînes de caractères static : Affichage et saisie d ’une chaîne - Exemple

```c
int main ()
{

    char name[50]; // Déclaration d'un tableau de caractères pour stocker le nom

    // Saisie du nom de l'utilisateur
    printf("Entrez votre nom : ");
    scanf("%s", name); // Utilisation de %s pour lire une chaîne de caractères

    // Affichage du nom saisi
    printf("Bonjour, %s !\n", name);


}
```

3 – Le Langage C -> Les fonctions

- Généralités
- Variables globales
- Variables locales
- Fonctions typées sans arguments
- Fonctions avec arguments
- Arguments par adresse
- Utilisation dans le programme
- Utilisation des entêtes .h
- Fonction main
- main : passage de paramètres

3 – Le Langage C -> Les fonctions : Généralités

- En C procédures = fonctions
- Pas de déclaration de fonction dans une autre fonction
- Possible appel d ’une fonction dans une autre fonction
- Variables locales
- Variables globales

3 – Le Langage C -> Les fonctions : La déclaration d'une fonction

```
type_de_retour Nom_de_la_Fonction
( type1 argument1, type2 argument2, ... )
{
        liste d'instructions ;
}
```

```
int add( int i, int j) {
        int k = i + j ;
        return k;
}
```

Déclaration de variable en dehors d'une fonction

- Eviter le surnombre
- Utilisation mots clé **extern**,

```
error: 'j' undeclared (first use in this function)
```

Exemple :

```c
#include <stdio.h>

int i = 100;

int main ()
{
    printf(" i = %d, j = %d",i,j);
}

int j = 200;
```

```c
#include <stdio.h>

int i = 100;

int main ()
{
    extern int j;
    printf(" i = %d, j = %d",i,j);
}

int j = 200;
```

3 – Le Langage C -> Les fonctions : Variables locales

- Variables locales aux fonctions : connues uniquement à l ’intérieur de la fonction qui les déclare

```
error: 'i' undeclared (first use in this function)

error: 'j' undeclared (first use in this function)
```

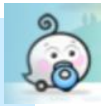
```
void FonctionTest() {
    int i =100;
    int j =200;
    printf(" i = %d, j = %d",i,j);
}


int main() {
    printf(" i = %d, j = %d",i,j);
}
```

3 – Le Langage C -> Les fonctions : Fonctions typées sans arguments

- Fonction renvoyant une valeur
- Type de la valeur déclaré avec la fonction
- Valeur retournée spécifiée grâce à retur

```
int  FonctionTestReturn() {
    int i =100;
    int j =200;
    return i + j;
}

int main() {
    int res = FonctionTestReturn();
    printf(" %d %d ", res , FonctionTestReturn());
}
```

3 – Le Langage C -> Les fonctions : Fonctions avec arguments



- Argument avec type void ou non
- Utilisation des valeurs de certaines variables du programme
- En-tête : définition de fonction

Fonctiontest.h

```
#ifndef FonctionTest
#define FonctionTest

void  FonctionTestArgument(int i, int j);
int  FonctionTestArgumentReturn(int i, int j);

#endif // FonctionTest
```

Fonctiontest.c

```
#include "FonctionTest.h"
void  FonctionTestArgument(int i, int j) {
    printf ("FonctionTestArgument : %d , %d \n ",i,j);
}
int  FonctionTestArgumentReturn(int i, int j) {
    FonctionTestArgument(i,j);
    return i + j;
}
```

```
FonctionTestArgument : 10000000 , 20000000
 FonctionTestArgument : 100 , 200
 FonctionTestArgument : 2 , 3
 main : 300 5
```

main.c

```
#include "FonctionTest.h"
int main() {
    FonctionTestArgument(10000000,20000000);
    int res = FonctionTestArgumentReturn(100,200);
    printf("main : %d %d \n", res , FonctionTestArgumentReturn(2,3));
}
```

3 – Le Langage C -> Les fonctions : Arguments par adresse

- Nécessaire pour modifier la variable
- Déclaration en-tête : utiliser les pointeurs
- Tableaux et *Pointeurs*

```c
#include <stdio.h>

int  FonctionCalul(int x,int y){
    y = 5 * x;
    return y;
}
void Calcul(int x,int y){
    y = 5 * x;
}
void CalculAdresse(int x,int* y){
    *y = 5 * x;
}
int main() {
    int i = 5;
    int j = 10;
    int res =0;
    printf ("A -> i= %d, j=%d, res=%d \n",i,j,res);
    res = FonctionCalul( i, j);
    printf ("B -> i= %d, j=%d, res=%d \n",i,j,res);
    Calcul(i,j);
    printf ("C -> i= %d, j=%d, res=%d \n",i,j,res);
    CalculAdresse(i,&j);
    printf ("D -> i= %d, j=%d, res=%d \n",i,j,res);

    return 0;
}
```

```
A -> i= 5, j=10, res=0
B -> i= 5, j=10, res=25
C -> i= 5, j=10, res=25
D -> i= 5, j=25, res=25
```

3 – Le Langage C -> Les fonctions : Utilisation des entêtes .

```
/*****************************/
/***     fichier: produit.h      ***/
/***   produit de 2 entiers    ***/
/*****************************/
#ifndef PRODUIT_H
#define PRODUIT_H
   int produit (int, int);

   int plus(int,int);
#endif  /* PRODUIT_H */
```

3 – Le Langage C -> Les fonctions : Utilisation des corps : .c

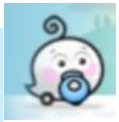
```
/*************************/
/***   fichier: produit.c   ***/
/***  produit de 2 entiers  ***/
/*************************/
#include « produit.h »
int produit (int a, int b)
 {
    return a*b;
 }
```

```
/*************************/
/***   fichier: plus.c   ***/
/***  produit de 2 entiers  ***/
/*************************/
#include « produit.h »
int plus(int a, int b)
 {
    return a+b;
 }
```

3 – Le Langage C -> Les fonctions : utilisation

```
#include <stdio.h>
#include "produit.h"

 int main () {
    int a, b,c;
    scanf("%d",&a);
   scanf("%d",&b);
   c = produit(a,b);
    printf("\nle produit vaut %d\n",c);
   return 0;
 }
```

3 – Le Langage C -> Les fonctions : Fonction main



```
int main() /* Plus petit programme C. */
{
    return 0;
}
```

La fonction main est une fonction spéciale, lorsqu'un programme est chargé, son exécution commence par l'appel de celle-ci, elle marque le début du programme.

Remarque :

Elle ne peut pas être récursive.

3 – Le Langage C -> Les fonctions : main : passage de paramètres

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

int main (int argc, char* argv [], char* envp[] )
{
  int i=0;

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

  printf("\n variables d'environnement.\n");
  for (i=0;envp[i] != NULL ;i++) printf("envp[%d]= %s \n",i,envp[i]);
  return 0;
}
```

3 – Le Langage C -> Les pointeurs

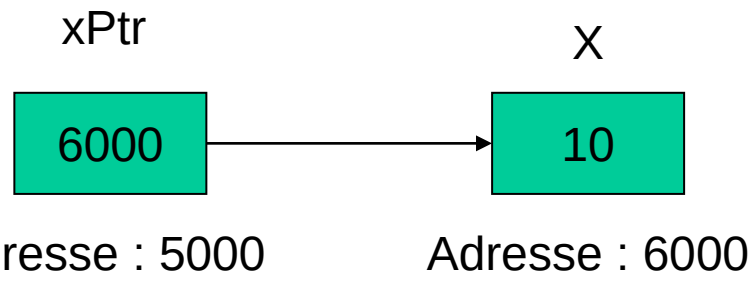
- Les pointeurs définitions
- Opérateur &
- Allocation dynamique de mémoire
- Opérateur de Déréférencement et d'indirection
- Allocation dynamique
- Libération de la mémoire
- Allocation dynamique en C
- Attention
- constants ou volatiles
- 6.11 - Fonction scanf
- 6.12 - Passage de paramètres par variable ou par valeur
- 6.13 - Pointeur de fonctions
- 6.14 - Conclusion

3 – Le Langage C -> Les pointeurs définition : Pointeur

- Tout emplacement mémoire a une adresse
- Un pointeur est une variable qui contient une adresse mémoire
- On dit que le pointeur pointe sur cette adresse

Déclaration pointeur
pointeur = type *** p**
Récupération valeur pointée
*** p**

```
int main ()
{
  int x = 10;
  int * xPtr;  /* xPtr pointeur pointant sur type int */
  xPtr = &x;
}
```



5000 6000 10
6000 10

3 – Le Langage C -> Les pointeurs définition : Adresse

Notion :

- Un objet manipulé est stocké dans sa mémoire.
- Une mémoire est constituée d'une série de « cases », ou sont stockées des valeurs (variables ou instructions)
- Pour accéder au contenu de la case mémoire où l'objet est enregistré, il faut connaître l'emplacement en mémoire de l'objet à manipuler.

Cet emplacement est appelé l'adresse de la case mémoire, ou l'adresse de la variable ou l'adresse de la fonction.

Toute case mémoire a une adresse unique.

int x = 10

X

10

Adresse : 6000

Notation adresse

adresse x = **&x**

3 – Le Langage C -> Les pointeurs : Opérateur &

- Retourne l ’adresse d’une variable en mémoire
- cf scanf

  – Exemple :

```
int i=8;
int* iptr = &i;
printf(‘‘Voici i et son adresse %d , %p‘‘,i,&i);
printf(‘‘Voici iptr et son adresse %p , %p, %d
   ‘‘,iptr,&iptr,*iptr);
*iptr=5;
printf(‘‘Voici i et son adresse %d , %p‘‘,i,&i);
printf(‘‘Voici iptr et son adresse %p , %p, %d
   ‘‘,iptr,&iptr,*iptr);
```

3 – Le Langage C -> Les pointeurs : Allocation dynamique de mémoire

L'allocation dynamique de mémoire permet de réserver de la mémoire (appelée encore *allocation*) pendant l'exécution d'un programme.

La quantité de mémoire utilisée par le programme est variable.

3 – Le Langage C -> Les pointeurs : Allocation dynamique de mémoire

- Réservation obligatoire pour pérennité
- Allocation par la fonction malloc
- Prototype dans <stdlib.h>

Exemples :

```
int i=0;
char * pc=(char*)malloc(10); //(char*) malloc(4*sizeof(char));
int* pi = (int*) malloc(4*sizeof(int));
for (i=0;i < 4 ;++i)
{
        pi[i]=i;
        printf("%d \n",pi[i]);
}
```

3 – Le Langage C -> Les pointeurs : Libération de la mémoire

- Fonction free(pointeur)
- Evite l'encombrement mémoire par la destruction des pointeurs

Exemples :

```
int i=0;
char * pc=(char*)malloc(10); //(char*) malloc(4*sizeof(char));
int* pi = (int*) malloc(4*sizeof(int));
for (i=0;i < 4 ;++i)
{
        pi[i]=i;
        printf("%d \n",pi[i]);
}
free(pi);
free(pc);
```

3 – Le Langage C -> Les pointeurs : Allocation dynamique en C

Allocation dynamique : malloc

- signature de la fonction :
  - void* malloc(unsigned int) ;
- Prototype dans alloc.h
- Exemples :
  - char* pc;
  - pc=(char*) malloc(10);
  - int * pi;
  - pi=(int*) malloc(sizeof(int)*16);

Libération mémoire : free

- Libération par la fonction
  - free(pointeur)
- Évite l'encombrement mémoire par la destruction des pointeurs
- Exemples :
  - free(pi);
  - free(pc);

3 – Le Langage C -> Les pointeurs : Attention

- Ne pas affecter directement une valeur dans un pointeur !
  - Exemple : **char *pc;**

    pc=0xfffe;

3 – Le Langage C -> Les pointeurs : constants ou volatiles

- **pi est un pointeur sur un entier constant.**

  int const *pi = (int*)malloc(sizeof(int));

  const int *pi = (int*)malloc(sizeof(int));

  *pi = 2; => erreur a la compilation

  pi = (int*)malloc(sizeof(int)); => OK a la compilation

- **pi est un pointeur constant sur un entier non constant**

  int j;

  int * const pj=&j;

  *pj = 2; => OK a la compilation

  pj = (int*)malloc(sizeof(int)); => erreur a la compilation

- **pi est un pointeur constant sur un entier constant**

  int j

  const int * const pc = &j;

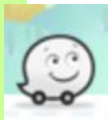
  *pi = 2; => erreur a la compilation

  pj = (int*)malloc(sizeof(int)); => erreur a la compilation

3 – Le Langage C -> Les pointeurs : Fonction scanf

- Utilisation : scanf(“%d“,&i);
- Le paramètre de la fonction est l’adresse de la variable
- Passage par adresse = modification possible du paramètre

3 – Le Langage C -> Les pointeurs : Conversions des tableaux en pointeurs



Accès aux éléments d'un tableau par pointeur

```
int tableau[100];
int *pi=tableau;

tableau[3]=5;     /* Le 4ème élément est initialisé à 5 */
pi[5]=1;          /* Le 6ème élément est initialisé à 1 */
```

3 – Le Langage C -> Les pointeurs : Pointeurs, tableaux

Déclaration d'un pointeur (adresse du premier élément) + allocation de l 'espace mémoire

On peut donc déclarer nous-même un pointeur et allouer la mémoire ' manuellement '

```
int *tab = (int*) malloc(sizeof(int)*5)
```

Avantage : allocation de l 'espace mémoire au moment de l 'exécution, attention ne pas oublier de détruire la mémoire:

```
free(tab) ;
```

Il est possible de pointer sur un élément particulier d'un tableau :

```
int tab[5];
int *ptr = &tab[2];    /* ptr pointe sur le 3eme élément du tableau */
```

Un nom de tableau utilisé dans une expression est converti en une adresse du début de ce tableau :

```
int tab[5];
int *ptr = tab;      /* équivalent a : "ptr = &tab[0]" */
```

3 – Le Langage C -> Les pointeurs : Les chaînes de caractères

- Une chaîne = un tableau de caractères
- Déclaration

```
char* nom;
nom=(char*)malloc(dim);
```

- Seulement dim-1 caractères disponibles
- Caractère NUL (‘ \0 ’) à la fin

3 – Le Langage C -> Les pointeurs : Les chaînes de caractères

Déclaration static

```
char texte[10]="bonjour";
printf("voici le texte %s\n",texte);
scanf("%9s",texte);
printf("voici le texte %s\n",texte);
```

Déclaration dynamique

```
#include <string.h>
int main() {

    char* texte = (char*)malloc(10);
    strcpy(texte,"bonjour");
    printf("voici le texte %s\n",texte);
    scanf("%9s",texte);
    printf("voici le texte %s\n",texte);
    free(texte);
    return 0;
}
```

3 – Le Langage C -> Les pointeurs : Conclusion

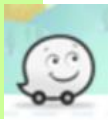
Les pointeurs sont très utilisés en C, mais ils sont très dangereux, quelques règles de survie :

Règle 1 : TOUJOURS INITIALISER LES POINTEURS QUE VOUS UTILISEZ.

Règle 2 : VÉRIFIEZ QUE TOUTE DEMANDE D'ALLOCATION MÉMOIRE A ÉTÉ SATISFAITE.

Règle 3 : LORSQU'ON UTILISE UN POINTEUR, IL FAUT VÉRIFIER S'IL EST VALIDE.

3 – Le Langage C -> Les types de variables complexes : Structures



- Types particuliers
- Composition d'un ensemble de types

  typedef struct {

  types; } nomdutype;

```
struct Nom_Structure {
        type_champ1 Nom_Champ1;

        type_champ2 Nom_Champ2;

        type_champ3 Nom_Champ3;

        type_champ4 Nom_Champ4;

        type_champ5 Nom_Champ5;

        ...
};
```

- Exemple

```
struct Etudiant {
    char nom[50];
    int age;
    float moyenne;
};
```

3 – Le Langage C -> Les types de variables complexes : Structures - Accéder aux champs

- Syntaxe : objet.champ

```c
int main() {
    // Déclaration de variables de type "Etudiant"
    struct Etudiant etudiant1;

    // Saisie des informations pour le premier étudiant
    printf("Entrez le nom de l'etudiant 1 : ");
    scanf("%s", etudiant1.nom);
    printf("Entrez la moyenne de l'etudiant 1 : ");
    scanf("%f", &etudiant1.moyenne);

    // Affichage des informations des étudiants
    printf("\nInformations de l'etudiant 1 :\n");
    printf("Nom : %s\n", etudiant1.nom);
    printf("Age : %d\n", etudiant1.age);
    printf("Moyenne : %.2f\n", etudiant1.moyenne);

    return 0;
}
```

- Pas de protection sur les champs

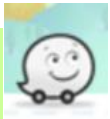
3 – Le Langage C -> Les types de variables complexes : Structures - Structures et pointeurs

- Pointeurs sur des structures
  - allocation dynamique de structures
  - modification d'une structure dans une fonction

```c
int main() {
    // Déclaration de variables de type "Etudiant"
    struct Etudiant* etudiant1 = (struct Etudiant*)malloc(sizeof(struct Etudiant));

    // Saisie des informations pour le premier étudiant
    printf("Entrez le nom de l'etudiant 1 : ");
    scanf("%s", (*etudiant1).nom);
    printf("Entrez l'age l'etudiant 1 : ");
    scanf("%f", & etudiant1->age);
    printf("Entrez la moyenne de l'etudiant 1 : ");
    scanf("%f", & etudiant1->moyenne);

    // Affichage des informations des étudiants
    printf("\nInformations de l'etudiant 1 :\n");
    printf("Nom : %s\n", etudiant1->nom);
    printf("Age : %d\n", (*etudiant1).age);
    printf("Moyenne : %.2f\n", etudiant1->moyenne);

    free(etudiant1);
    return 0;
}
```

- Notation équivalente : ->
  - Exemple : **(*etudiant1).age** **etudiant1->age**

3 – Le Langage C -> Les types de variables complexes enum



Syntaxe :

enum [<nom de l'énumération>] {<nom d'une constante> [= <valeur>], ...} [<liste de variables>];

Description:

Le mot clé enum permet de définir un ensemble de constantes de type entier(int).
Une énumération fournit des identificateurs mnémoniques pour un ensemble de valeurs entières et finies
Une variable énumérée ne peut se voir affecter qu'un de ses énumérateurs(<constante>).
En l'absence d'une <valeur>, la première constante prend la valeur zéro.
Toute constante sans <valeur> sera augmentée d'un par rapport à la constante précédente.

Exemple :

```
enum jour { Lundi, Mardi, Mercredi, Jeudi, Vendredi, Samedi, Dimanche};
enum jour { Lundi, Mardi, Mercredi, Jeudi, Vendredi, Samedi, Dimanche} variable_jour;
enum { Lundi=1, Mardi=2, Mercredi=3, Jeudi=4, Vendredi=5, Samedi=6, Dimanche=7};
```

3 – Le Langage C -> Les types de variables complexes enum

```c
enum JourSemaine {
    LUNDI, MARDI, MERCREDI, JEUDI, VENDREDI, SAMEDI, DIMANCHE
};

int main() {
    // Déclaration de variables de type "enum JourSemaine"
    enum JourSemaine jour1;
    // Attribution de valeurs aux variables
    jour1 = LUNDI;
    // Affichage des jours de la semaine
    printf("Jour 1 : %d \n",jour1);
    switch (jour1) {
        case LUNDI:
            printf("Lundi\n");
            break;
        case MARDI:
            printf("Mardi\n");
            break;
        case MERCREDI:
            printf("Mercredi\n");
            break;
        case JEUDI:
            printf("Jeudi\n");
            break;
        case VENDREDI:
            printf("Vendredi\n");
            break;
        case SAMEDI:
            printf("Samedi\n");
            break;
        case DIMANCHE:
            printf("Dimanche\n");
            break;
        default:
            printf("Jour inconnu\n");
    }
    return 0;
}
```

3 – Le Langage C -> Les types de variables complexes Types synonymes : typedef

- Création de ses propres types
- Déclaration en début de programme

- Exemple
  - **typedef int entier; /* type entier = int */**
  - **typedef int Type_tab[5]; /* type Type_tab = tableaux de 5 entiers */**
  - **typedef char* fcar; /* type fcar = pointeur de char */**
  - **typedef int* vecteurInt;**
  - **typedef vecteurInt* matriceInt;**

3 – Le Langage C -> Les types de variables complexes Types synonymes : typedef

```c
#include <stdio.h>

// Définition de la structure "Etudiant"
typedef struct etudiant {
    char nom[50];
    int age;
    float moyenne;
} Etudiant;

int main() {
    // Déclaration de variables de type "Etudiant"
    Etudiant* etudiant1 = (Etudiant*)malloc(sizeof(Etudiant));
    // avant struct Etudiant* etudiant1 = (struct Etudiant*)malloc(sizeof(struct Etudiant));

    // Saisie des informations pour le premier étudiant
    printf("Entrez le nom de l'etudiant 1 : ");
    scanf("%s", (*etudiant1).nom);
    printf("Entrez l'age l'etudiant 1 : ");
    scanf("%f", & etudiant1->age);
    printf("Entrez la moyenne de l'etudiant 1 : ");
    scanf("%f", & etudiant1->moyenne);

    // Affichage des informations des étudiants
    printf("\nInformations de l'etudiant 1 :\n");
    printf("Nom : %s\n", etudiant1->nom);
    printf("Age : %d\n", (*etudiant1).age);
    printf("Moyenne : %.2f\n", etudiant1->moyenne);

    free(etudiant1);
    return 0;
}
```

3 – Le Langage C -> Les types de variables complexes Types synonymes : typedef

```c
// Définition de l'énumération "JourSemaine"
typedef enum jourSemaine {
    LUNDI, MARDI, MERCREDI, JEUDI, VENDREDI, SAMEDI, DIMANCHE
} JourSemaine;

int main() {
    // Déclaration de variables de type "enum JourSemaine"
    JourSemaine jour1; // avant enum JourSemaine jour1;
    // Attribution de valeurs aux variables
    jour1 = LUNDI;
    // Affichage des jours de la semaine
    printf("Jour 1 : %d \n",jour1);
    switch (jour1) {
        case LUNDI:
            printf("Lundi\n");
            break;
        case MARDI:
            printf("Mardi\n");
            break;
        case MERCREDI:
            printf("Mercredi\n");
            break;
        case JEUDI:
            printf("Jeudi\n");
            break;
        case VENDREDI:
            printf("Vendredi\n");
            break;
        case SAMEDI:
            printf("Samedi\n");
            break;
        case DIMANCHE:
            printf("Dimanche\n");
            break;
        default:
            printf("Jour inconnu\n");
    }
    return 0;
}
```

4 - Bibliographie

- **Le langage C : Norme ANSI**
  - **Brian-W Kernighan et Dennis-M Ritchie**
- **Langage C**
  - **Claude Delannoy**
- **Méthodologie de la programmation en C : Norme C 99 - API POSIX**
  - **Achille Braquelaire**