

Alphorm.com

# Langage C

Site : <http://www.alphorm.com>  
Blog : <http://blog.alphorm.com>

Rabah ATTIK

Formateur et Consultant indépendant  
Ingénierie système embarqué

Formation Le langage C

alphorm.com™©

## C Plan

- Présentation du formateur
- Présentation de la formation
- Les références bibliographiques
- Autres ressources utiles



Formation Le langage C

alphorm.com™©

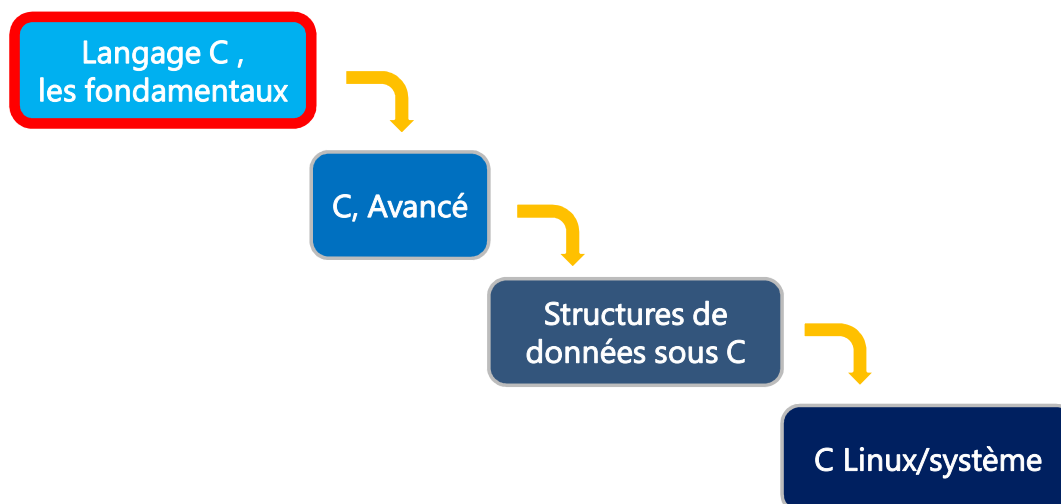
## C Présentation du formateur

### Rabah ATTIK

- Email: [rabah.attik@inge-tech.com](mailto:rabah.attik@inge-tech.com)
- Ingénieur en mécanique
  - Actuellement consultant système embarqué
- Mission de développement bas niveau
- Mes références :
  - Profil LinkedIn : <https://www.linkedin.com/in/rabah-attik-49631714>
  - Profil Viadeo : <http://fr.viadeo.com/fr/profile/rabah.attik>



## C Cours Langage C sur Alphorm



## **C** Présentation de la formation

- Le langage C
- Chapitre 2: Premiers pas en C
- Chapitre 3: La mémoire et le programme
- Chapitre 4: Types, opérateurs et expressions
- Chapitre 5: Structures de contrôle
- Chapitre 6: Pointeurs, tableaux et chaînes de caractères
- Chapitre 7: Les types structurés
- Chapitre 8: Les fonctions
- Chapitre 9: Compilation séparée
- Chapitre 10: Le préprocesseur
- Chapitre 11: La bibliothèque standard



## **C** Les références bibliographiques

- **Le langage C / Norme ANSI** (auteur, K et R)
  - Traduit du livre original
- **Le guide complet du langage C** (Claude Delannoy)
  - Les concepts sont détaillés

## C Autres ressources utiles

---

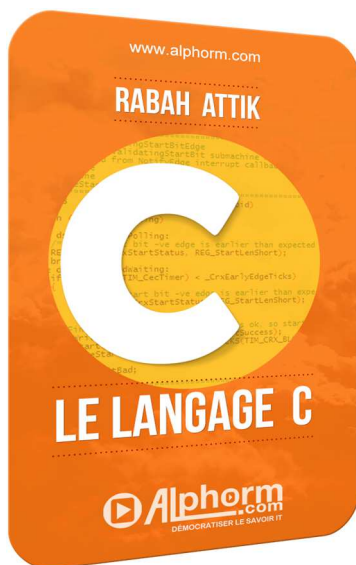
- [https://fr.wikibooks.org/wiki/Programmation\\_C](https://fr.wikibooks.org/wiki/Programmation_C)
- [http://www.tutorialspoint.com/c\\_standard\\_library/](http://www.tutorialspoint.com/c_standard_library/)

## C

---

LE LANGAGE C, VOUS APPELLE!





**Alphorm**.com

## Présentation de la formation

---

### Préparer l'environnement de développement

Site : <http://formation.inge-tech.com>  
Blog : <http://www.alphorm.com/blog>  
Forum : <http://www.alphorm.com/forum>

**Rabah ATTIK**

Formateur et consultant indépendant  
Ingénierie système embarqué  
rabah.attik@inge-tech.com

Formation Le langage C

alphorm.com™©

## C Plan

---

- Windows
  - Mise en place de Eclipse
  - MinGW
- Virtual Box
  - Linux: vim / gcc



Formation Le langage C

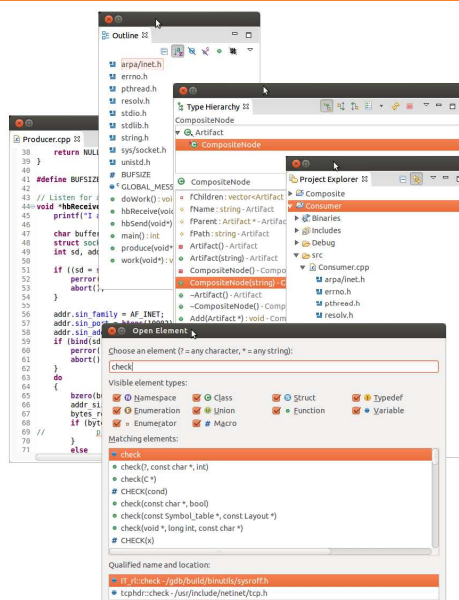
alphorm.com™©

## C Windows

- Télécharger Eclipse CDT
- Télécharger MinGW
- Configurer Eclipse

## C Linux

- Mise en place d'un machine virtuelle  
« ubuntu 12.04 »
- Installer vim



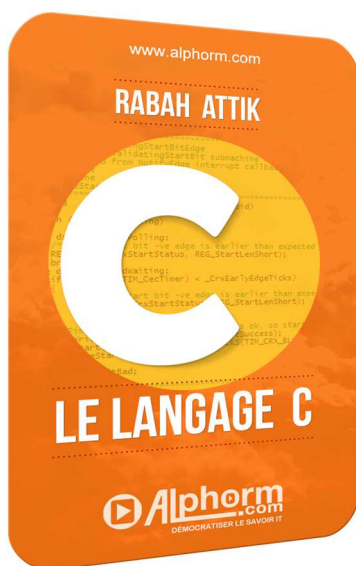
## C Ce qu'on a couvert

- Mise en place de l'environnement de développement
  - Cas windows
  - Cas Linux en machine virtuelle



Formation Le langage C

alphorm.com™©



**Alphorm**.com

## Premier pas en C

### Présentation du langage et ses atouts

Site : <http://www.alphorm.com>  
Blog : <http://blog.alphorm.com>

**Rabah ATTIK**

Formateur et Consultant indépendant  
Ingénierie système embarqué

Formation Le langage C

alphorm.com™©

## C Plan

---

- Un peu d'histoire
- Le langage C à sa création
- L'évolution du langage C



## C Un peu d'histoire

---

- Avant le langage C, le contexte
- La création du langage C
  - Créé en 1972 par **Denis Ritchie**
  - Objectif limité, écrire un système d'exploitation (UNIX)
- Premières « normalisations »
- Ecrits en C: Unix / GCC / Noyau linux / GNOME





## C Le langage C à sa création

---

- Qualité opérationnelle
- Syntaxe riche et typée
- Très rapidement adopté et normé

## C L'évolution du langage C

---

- C K & R jusqu'à C11
- Les caractéristiques du C
  - Impératif, généraliste, simple
  - Langage de bas niveau
  - Supporte les types énumérés, composés, opaques
  - Non gérés: objets de plus haut niveau, programmation objet, exception, pas d'espace de noms
  - Les grandes qualités et les défauts cités

## C Ce qu'on a couvert

---

- Rappel historique
- Atouts du C
- Caractéristiques du C



**Alphorm**.com

Premiers pas en C

---

Un premier  
programme

Site : <http://www.alphorm.com>  
Blog : <http://blog.alphorm.com>

**Rabah ATTIK**

Formateur et Consultant indépendant  
Ingénierie système embarqué

## C Plan

- Présentation d'un programme basique
  - Structure
  - Syntaxe



## C

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

int main(void) {
    // Déclaration de variables
    int entier, carreEntier;

    // Présentation du programme
    printf("Bonjour, ce programme calcul le carré d'un nombre entier\n");
    printf("Veuillez entrer votre nombre entier\n");

    // Saisi d'un nombre entier et stockage dans la variable 'entier'
    scanf("%d",&entier);
    printf("Vous avez entrer le nombre: %d\n", entier);

    //Calcul du carré
    carreEntier = entier * entier;
    printf("Le carré du nom %d est: %d",entier, carreEntier);

    return EXIT_SUCCESS;
}
```

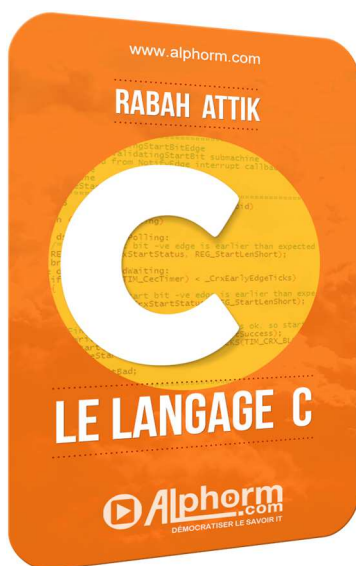
## C Ce qu'on a couvert

- Aperçu d'un programme sur l'IDE éclipse
- Structure
- Syntaxe



Formation Le langage C

alphorm.com™©



**Alphorm**.com

## Premiers pas en C

### Utilisation élémentaire de la chaîne de production

Site : <http://www.alphorm.com>  
Blog : <http://blog.alphorm.com>

**Rabah ATTIK**

Formateur et Consultant indépendant  
Ingénierie système embarqué

Formation Le langage C

alphorm.com™©

## C Plan

---

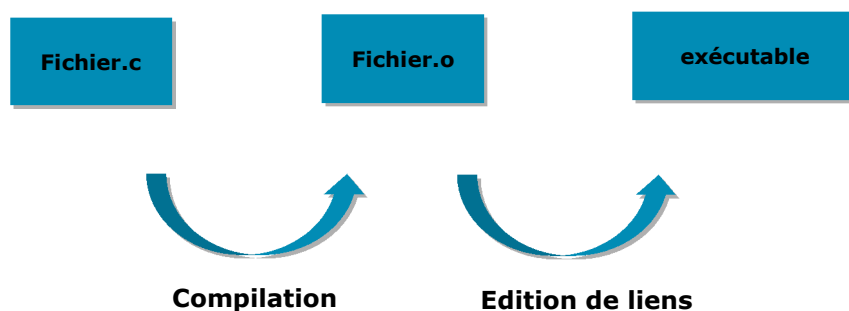
- Du fichier source au binaire
- Le compilateur
- Lancement du programme



## C Du fichier source au binaire

---

- Chaîne de production



## **C** Du fichier source au binaire

- Le compilateur
  - Traduction du code source en langage C en langage machine
  - Générer un fichier objet
- Les étapes
  - Analyse lexicale, prétraitement, analyse syntaxique

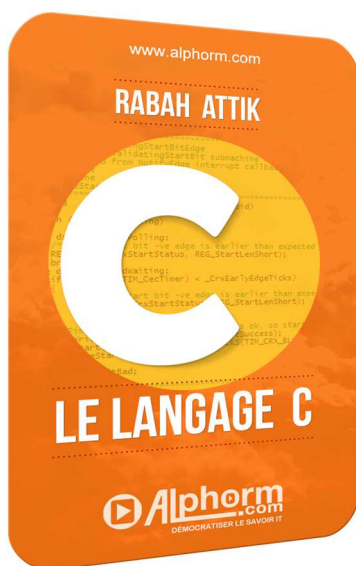
## **C** Lancement du programme

- Compilation du programme de démo
- Lancement du programme de démo

## C Ce qu'on a couvert

---

- La chaine élémentaire de production
- Le compilateur
- Cas concret



**Alphorm**.com

Premiers pas en C

---

Les outils  
de développement

Site : <http://www.alphorm.com>  
Blog : <http://blog.alphorm.com>

Rabah ATTIK

Formateur et Consultant indépendant  
Ingénierie système embarqué

## C Plan

---

- Les éditeurs connus
- Les compilateurs
- Les IDE (*Integrated Development Environment*)



## C Les éditeurs

---

- Utilité = écrire du code
- Les options attendues:
  - Code formaté lisible : coloration syntaxique
  - Fonctionnelle: auto-completion, repliage de code, indentation automatique,...
- vim / emacs / gedit / notepad++ / blocnote windows



## **C** Les compilateurs

---

- Compilateurs et OS
- gcc
- minGW
- Cl, le compilateur Windows natif utilisé par visual studio
- Digital Mars C/C++ Free Compiler
- Borland Compiler 5.5
- ICC
- La problématique processeur Intel vs AMD

## **C** Les IDE

---

- Eclipse
- Visual studio (MSVC)
- Netbeans
- Code::Blocks
- Borland C++
- Dev-C++
- xCode
- Kdevelop

## C Les IDE

---

- Eclipse



- Leaders: Borland, IBM, MERANT, QNX Software Systems, Rational Software, Red Hat, SuSE
- 2001, 1<sup>er</sup> version and 2004
- Plus de 250 projets open source
- Plugins
- Disponible sous windows / Linux / Mac OS
- <https://eclipse.org/downloads/>

## C Les IDE

---

- Visual studio (MSVC)



- Licence propriétaire
- Dédié pour windows
- <https://msdn.microsoft.com/fr-fr/vstudio/>

## **C** Les IDE

---

- Netbeans
  - Créé par Sun en juin 2000
  - OpenSource
  - Licence CDDL et GPLv2
  - Plugins
  - Disponible sous Windows / Linux / Mac OS
  - <https://netbeans.org/downloads/>



## **C** Les IDE

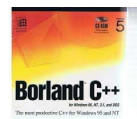
---

- Code::Blocks
  - Codé avec le frameworks wxWidgets
  - Opensource / crossplateforme
  - Dédié au développement C/C++ et fortran
  - Simple intuitif
  - Plugins
  - Supporte plusieurs compilateurs ()
  - Importation de projets d'autres IDE
  - [www.codeblocks.org/](http://www.codeblocks.org/)



## C Les IDE

- Borland C++
  - Windows
  - Licence propriétaire
  - Développé par Borland
  - Spécifique pour C/C++
  - Turbo C++ → **Borland C++** → Borland C++Builder → CodeGear C++Builder → [EmbarcaderoC++Builder](#)
  - <http://www.borland.com/Home>



## C Les IDE

- Dev-C++
  - Windows
  - Licence GNU GPL
  - Développé avec Borland Delphi 6
  - Spécifique C/C++
  - Utilise la version MinGW de GCC
  - Gestion des classes et des fonctions et débogueur avec suivi des variables
  - Gestion des ressources néant
  - <http://orwelldevcpp.blogspot.com>



## C Les IDE

---

- xCode
  - Carbon (C / C++) Cocoa (Objective-C / AppleScript / Java / Swift)
  - Mac OS X et iOS
  - Licence propriétaire
  - <http://developer.apple.com/technologies/tools/xcode.html>



## C Les IDE

---

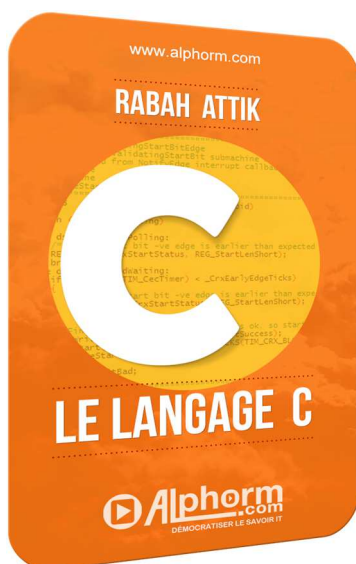
- Kdevelop
  - IDE sous linux
  - Très nombreuses features ( affichage d'info sur le code / context / debugging / completion )
  - Cmake
  - Svn
  - C / C++ / php
  - En projet: test unitaire / valgrind / support QtDesigner / php debugger / Java C# XML et CSS
  - Licence GPL
  - [www.kdevelop.org](http://www.kdevelop.org)



## C Ce qu'on a couvert

---

- Editeurs
- Compilateurs
- IDE



**Alphorm**.com

## Premiers pas en C

---

## Exécution du premier programme

Site : <http://www.alphorm.com>  
Blog : <http://blog.alphorm.com>

**Rabah ATTIK**

Formateur et Consultant indépendant  
Ingénierie système embarqué

## C Plan

- Exécution de notre premier programme
- Choix de IDE : Eclipse
- Choix de l'OS : Windows
- Compilateur : MinGW



## C Programme à exécuter

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

int main(void) {
    // Déclaration de variables
    int entier, carreEntier;

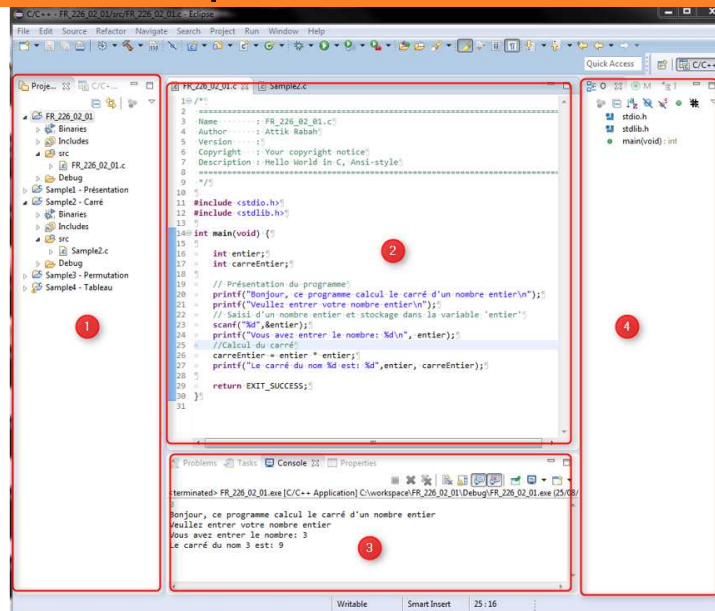
    // Présentation du programme
    printf("Bonjour, ce programme calcul le carré d'un nombre entier\n");
    printf("Veuillez entrer votre nombre entier\n");

    // Saisi d'un nombre entier et stockage dans la variable 'entier'
    scanf("%d",&entier);
    printf("Vous avez entrer le nombre: %d\n", entier);

    //Calcul du carré
    carreEntier = entier * entier;
    printf("Le carré du nom %d est: %d",entier, carreEntier);

    return EXIT_SUCCESS;
}
```

## C L'IDE eclipse



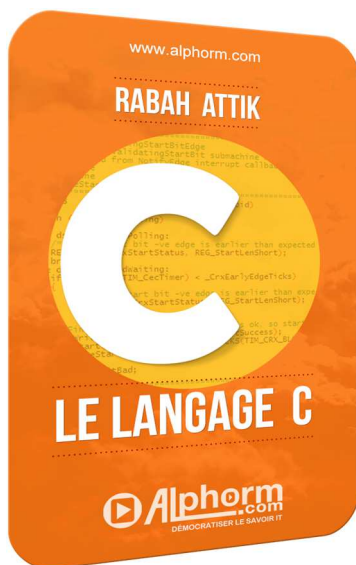
1. Fenêtre projet
2. Fenêtre fichier
3. Fenêtre console
4. Fenêtre outline

## C Ce qu'on a couvert

- Exécution du programme
- Choix de l'IDE et démonstration







**Alphorm**.com

Les mémoires et le programme

## RAM et autres mémoires

Site : <http://www.alphorm.com>  
Blog : <http://blog.alphorm.com>

Rabah ATTIK

Formateur et Consultant indépendant  
Ingénierie système embarqué

Formation Le langage C

alphorm.com™©

### C Plan

- Définition de la mémoire
- La mémoire vive (RAM random access memory), la mémoire dynamique
- La mémoire morte (ROM read only memory)
- Les autres ROM
- La mémoire de masse (disques durs)
- Les mémoires flash
- Les mémoires rapides et ultra rapides (registre et cache)



Formation Le langage C

alphorm.com™©

## **C** Définition de mémoire

---

- Dispositif électronique qui sert à stocker des informations
- Rémanence: persistance des données en cas de coupure de l'énergie
- Caractéristiques techniques:
  - Capacité
  - Temps d'accès
  - Temps de cycle
  - Débit
  - Non volatilité

## **C** La mémoire vive

---

- Statique SRAM
- Dynamique DRAM et SDRAM

## C La mémoire vive

---

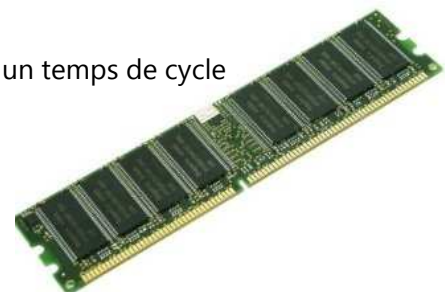
- La SRAM (mémoire statique), principe
  - Bascule
  - Plus volumineuse
  - Rapide
  - Chère
  - Cache processeur



## C La mémoire vive

---

- La DRAM (mémoire dynamique), principe
  - Centaines de milliers de condensateurs, chargé = 1 (représente un bit)
  - Cycle de rafraichissement (statique vs dynamique)
  - Couplé à un transistor (pour récupérer ou modifier l'état du condensateur)
  - Rangés sous forme d'une matrice => une case mémoire est accessible par ligne et colonne.
    - Cet accès se fait après un temps de latence et après un temps de cycle
  - Synchrones vs Asynchrones



## C La mémoire vive

---

- La mémoire dynamique asynchrone
  - les anciennes DRAM
    - La DRAM FPM (Fast Page Mode, 1987)**  
Temps d'accès ~70 à 80 ns. Accès rapide sur une même page mémoire. La fréquence de ce type de mémoire variait de 33 à 50 Mhz.
    - La DRAM EDO (Extended Data Out, 1995)**  
La mémoire EDO apportent quelques améliorations par rapport à la FPM. Temps d'accès réduit à 50 ou 60 ns. On peut notamment accéder aux données pendant les périodes de rafraîchissement.  
La fréquence de ce type de mémoire variait de 33 à 50 Mhz.
    - La DRAM BEDO (Burst EDO)**  
La mémoire BEDO permet le transfert des données en **mode rafale**. Les données sont envoyées par flux ininterrompu. La fréquence de ce type de mémoire était de 66 Mhz.

## C La mémoire vive

---

- La mémoire dynamique synchrone :
  - SDRAM, 1997
  - RDRAM DRDRAM, 1999
  - DDR-I, 2000
  - DDR-II, 2004
  - DDR-III, 2007
  - DDR4 SDRAM, 2015 (?)

## C La mémoire vive

- La mémoire dynamique, les connecteurs

Il existe différents types de connecteurs en fonction des barrettes mémoires utilisées :

**SIMM** (Single In-line Memory Module) : Début 80

Contact d'un côté, deux versions:

- 30 connecteurs pour les bus 16 bits (aucun détrompeur)
- 72 connecteurs pour les bus 32 bits (1 détrompeur)

- **DIMM** (Dual In-line Memory Module) : ~1997

Successeur du SIMM permettant de transférer 64 bits. Le connecteur DIMM se décline en 2 versions :

- 168 broches pour la SDRAM (2 détrompeurs)
- 184 broches pour la DDR-SDRAM (1 détrompeur)

- **SO-DIMM** (Small Outline DIMM) : (version compacte DIMM, utilisée pour les ordinateurs portables)

- 32 bits, 72 connecteurs
- 64 bits, 144 connecteurs

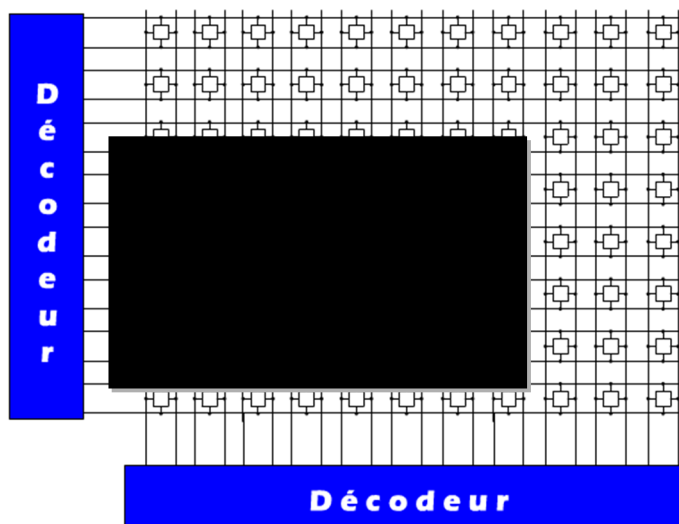
- **RIMM** (Rambus In-line Memory Module) :

Format spécifique aux barrettes Rambus : 184 connecteurs (2 détrompeurs)

- **Exemple** : Une barrette mémoire DIMM DDR3 comporte 240 contacts. Les SO-DIMM DDR3, destinées aux ordinateurs portables, ont quant à elles 204 contacts

## C La mémoire vive

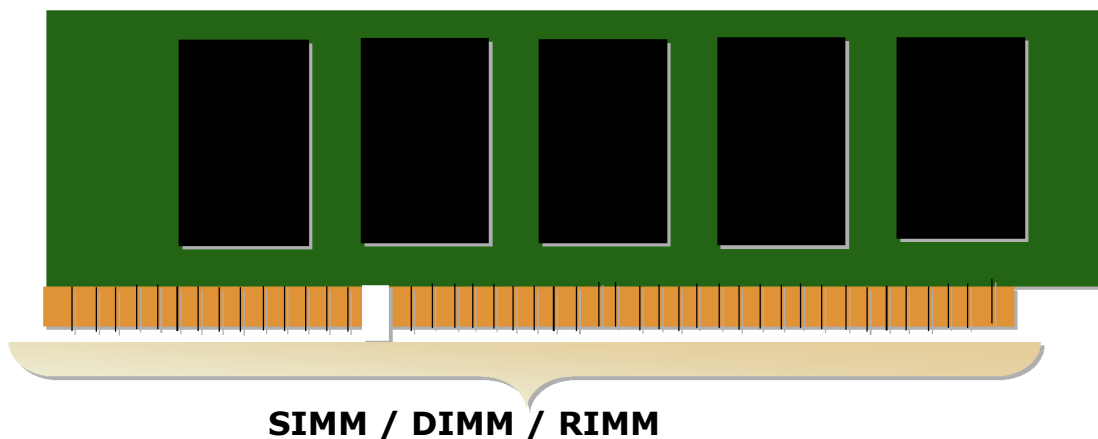
- Cellule
  - 1bit



## C La mémoire vive

- Dynamique RAM

### Barette mémoire



## C La mémoire vive

- Dynamique RAM

Evolution des DRAM

Année	Désignation	Type	Largeur Bus
1987	Fast Page Mode (FPM)	Asynchronous	4 ou 8 octets
1990	Extended Data Out (EDO)	Asynchronous	4 ou 8 octets
1997	SDRAM PC 66	Synchronous	8 octets
1998	SDRAM PC 100	Synchronous	8 octets
1999	SDRAM PC 133	Synchronous	8 octets
1999	Rambus PC600/700/800	Synchronous	2 octets
2000	DDR-SDRAM PC 1600	Synchronous	8 octets
2001	DDR-SDRAM PC 2100	Synchronous	8 octets
2002	DDR-SDRAM PC 2700/3200	Synchronous	8 octets
2002	Rambus PC 1066	Synchronous	2 octets
2003	DDR-SDRAM PC 3500/4000/4400	Synchronous	8 octets
2003	DDR2-SDRAM PC 4200	Synchronous	8 octets
2007	DDR3	Synchronous	8 octets

## C La mémoire vive

---

- Dynamique RAM



## C La mémoire vive

---

- Conclusion RAM
  - Mémoire statique à bascule
  - Très rapide (quelques ns)
  - Volumineux
  - Consomme de l'énergie
  - Chère
  - Mémoire dynamique à transistor couplé à un condensateur
  - Rafrachissement
  - Temps d'accès long
  - Moins chère
  - Haute densité

La MRAM ?

<http://www.cnetfrance.fr/news/la-mram-pourrait-remplacer-la-dram-a-partir-de-2018-39795802.htm>

## C ROM

---

- Qu'est ce que la ROM?
  - Les mémoires mortes ou ROM sont non volatiles et vouées à être accédées en lecture seule
  - Contiennent des données (code ou données) non amenées à changer souvent
  - Routine d'accès de base des périphériques

## C Les autres ROM

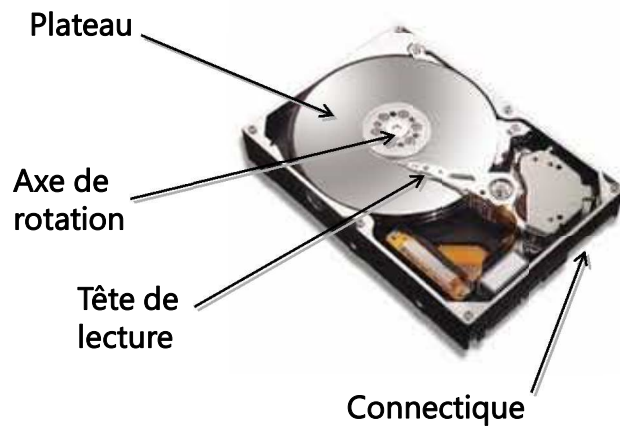
---

- PROM (*Programmable ROM*)
- EPROM (*Erasable programmable ROM*)
- EEPROM (*Electrically Erasable Programmable ROM*)



## C Mémoire de masse

- Disque dur traditionnel
- IDE
- eSata



## C Mémoire flash

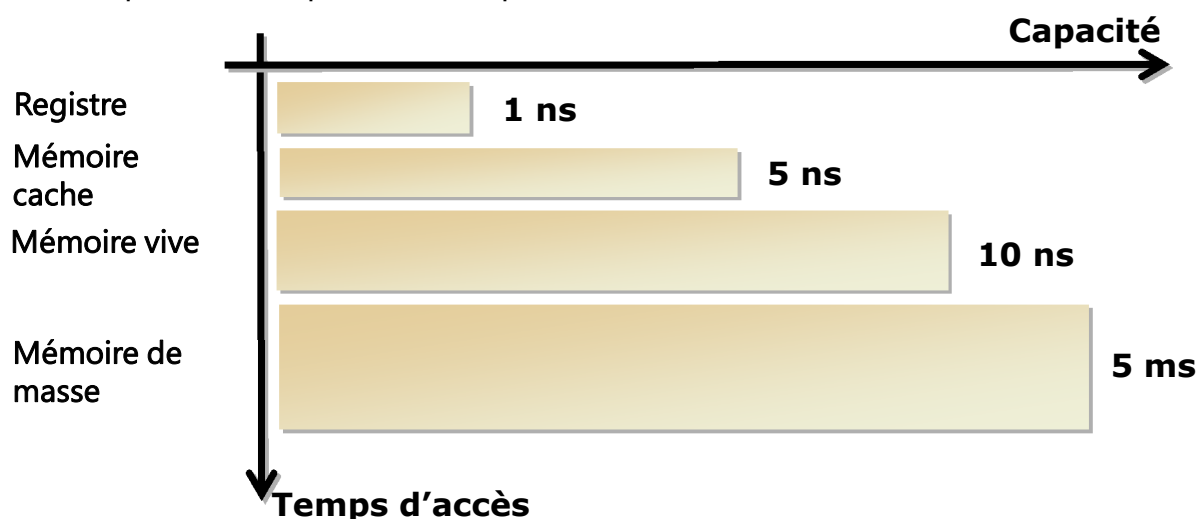
- Flash = synthèse entre EEPROM et SRAM
- Mémoire de masse sans disque
- Alternative aux disques durs

## C Mémoire rapide et ultra rapide

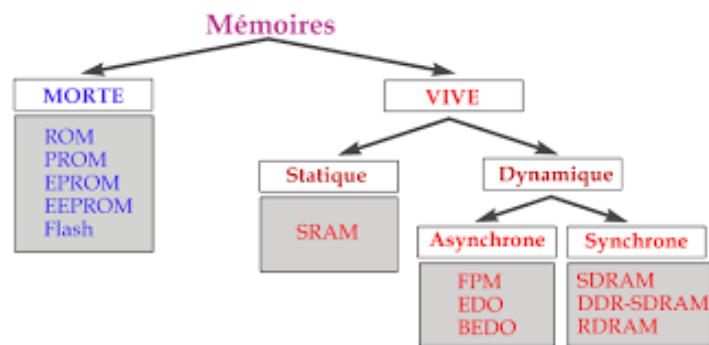
- Registre
  - Petites mémoires pour stocker les informations pour le processeur
    - Registre d'état / compteur ordinal / registre tampon
- Cache
  - Mémoire rapide
  - Réduire délai d'attente entre la mémoire vive et le processeur
  - 3 niveaux

## C Comparaison

- Comparaison Capacité / Temps d'accès



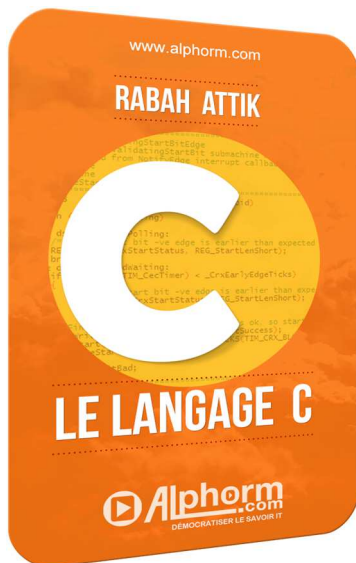
## C Récapitulatif



## C Ce qu'on a couvert

- RAM
- ROM
- Disque dur
- Mémoire ultra rapide
- Mémoire flash





**Alphorm**.com

Les mémoires et le programme

## Gestion d'un programme par l'OS

Site : <http://www.alphorm.com>  
Blog : <http://blog.alphorm.com>

Rabah ATTIK

Formateur et Consultant indépendant  
Ingénierie système embarqué

Formation Le langage C

alphorm.com™©

## C Plan

- Programme et processus
- L'instanciation en RAM
- Exécution en RAM



Formation Le langage C

alphorm.com™©

## C Programme et processus

---

- Le processus = l'exécution du programme
- S'exécuter, c'est d'abord exister en RAM

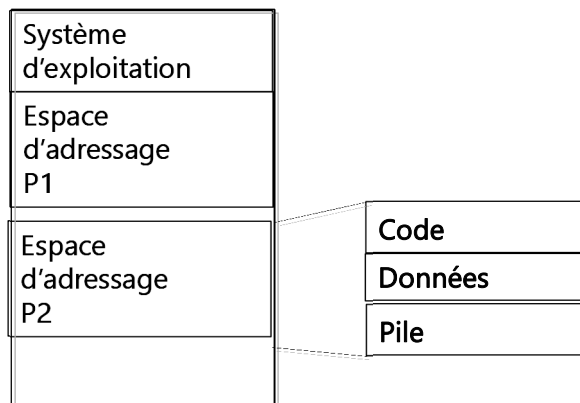
## C Instanciation en RAM

---

- Les structures réservées par l'OS à l'instanciation d'un programme
- La pagination et la segmentation

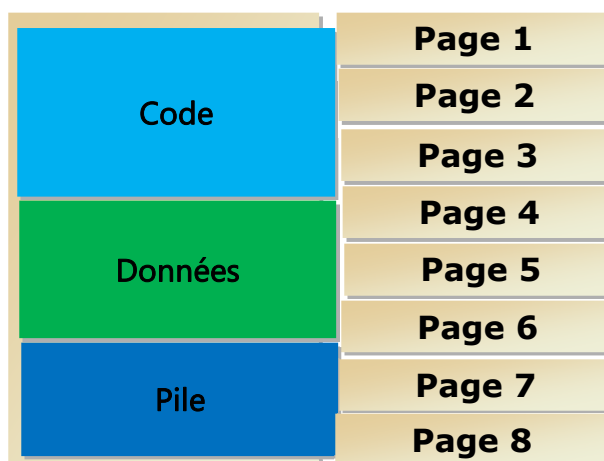
## C Instanciation en RAM

- Les structures réservées par l'OS à l'instanciation d'un programme
- 3 espaces de tailles différentes :
  - Segments
    - Code
    - Données
    - Pile



## C Instanciation en RAM

- La pagination et la segmentation
  - Les segments sont paginés



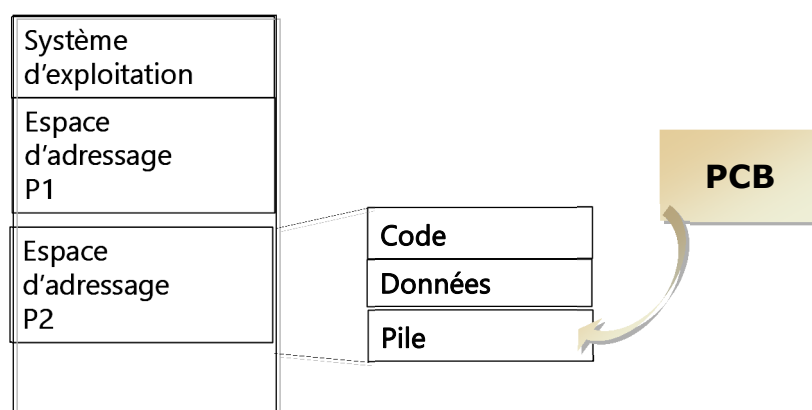
## C Instanciation en RAM

- La pagination et la segmentation
  - Les segments sont paginés
  - La mémoire est structurée en case
  - Les pages d'un segment sont chargés dans les cases libres de la mémoire centrale

<b>Case 1</b>
<b>Case 2    Page 1</b>
<b>Case 3</b>
<b>Case 4    Page 2</b>
<b>Case 5</b>
<b>Case 6    Page 3</b>
<b>Case 7</b>

## C Exécution du programme

- PCB et état d'un processus



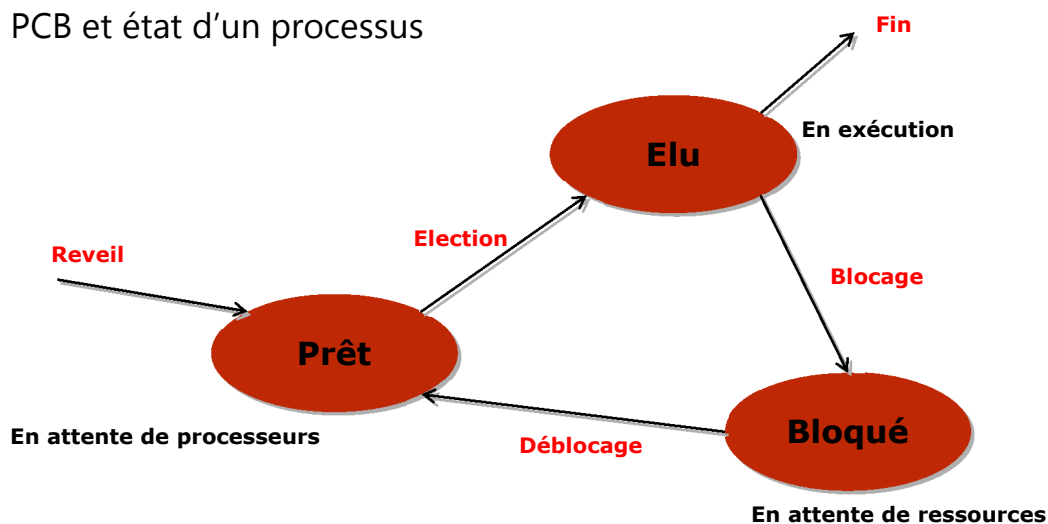
## C Exécution du programme

- PCB et état d'un processus

Identificateur processus
Etat du processus
Contexte pour reprise (registre et pointeurs, piles, compteur instruction, ...)
Pointeurs sur file d'attente et priorité (ordonnancement)
Informations mémoire (limite et tables pages/segments)
Informations de comptabilisation et sur les E/S, périphériques aloués, fichiers ouverts, ...

## C Exécution du programme

- PCB et état d'un processus





## C Exécution du programme

---

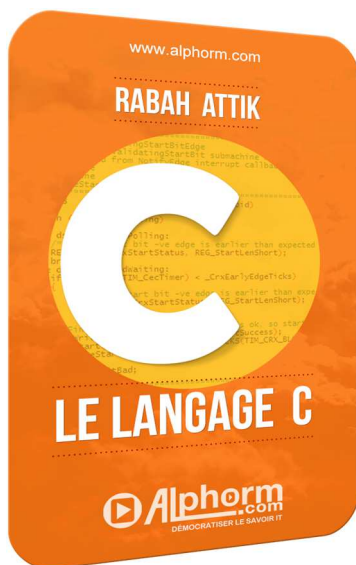
- Les bus d'adresses et de données
- Les instructions et le processeurs
- Les cas limites et de plantage

## C Ce qu'on a couvert

---

- Programme et processus
- Instanciation en RAM
- Exécution en RAM





**Alphorm**.com

## Types, opérateurs et expressions

### Types et opérateurs arithmétiques

Site : <http://www.alphorm.com>  
Blog : <http://blog.alphorm.com>

Rabah ATTIK

Formateur et Consultant indépendant  
Ingénierie système embarqué

Formation Le langage C

alphorm.com™©

## C Plan

- Notion de type
- Types entiers
- Types flottants
- Types caractères
- Initialisation de types et constantes
- Opérateurs arithmétiques
- Propriétés relatives des opérateurs




Formation Le langage C

alphorm.com™©

## C Notion de type

- Le C un langage typé
- Type scalaire (ou simple) vs type structuré (ou agrégé)
- Représenter une information (un nombre ou un caractère ASCII) grâce au type
- Type de base :
  - Type entier déclaré avec le mot clé « int »
  - Type flottant déclaré avec le mot clé « float »
  - Type caractère déclaré avec le mot clé « char »

## C Types entiers

- Mot clé: « int »
- Entier relatif sur (n-1) bit et un bit pour le signe
- Trois tailles différentes: « *short int* », « *int* » et « *long int* »
-  *dépend des machines*
- Nuancé par « *unsigned* » => n bits pour la valeur
- Exemple, sur un PC processeur 32 bits: short int et int occupe 16 bits soit 2 octets et long int occupe 32 bits soit 4 octets
  - « int » : -32768 à 32767
  - « unsigned int »: 0 à 65535
- Pour introduire une constante entière, il suffit de l'écrire: « +505 », « -404 »

## C Types flottants

- Représentation virgule fixe:
  - Représenter la partie entière et la partie décimale en binaire
- **Exemple:** 28.247 : 28 partie entière et 0.247 partie décimale
  - $28 = 4+8+16 \Leftrightarrow 11100$
  - 0.247: On cherche des puissance négatives en multipliant par 2

$$\begin{aligned}
 0.247 * 2 &= 0.494 < 1 \Leftrightarrow 0 * 2^{-1} \\
 0.494 * 2 &= 0.988 < 1 \Leftrightarrow 0 * 2^{-2} \\
 0.988 * 2 &= 1.976 > 1 \Leftrightarrow 1 * 2^{-3} \\
 0.976 * 2 &= 1.952 > 1 \Leftrightarrow 1 * 2^{-4} \\
 0.952 * 2 &= 1.904 > 1 \Leftrightarrow 1 * 2^{-5} \\
 &\dots
 \end{aligned}
 \qquad
 0.247 \Leftrightarrow 00111...$$

$$28.247 \Leftrightarrow 11100,00111...$$

## C Types flottants

- Le C se base sur la notation exponentielle:  $M.B^E \Rightarrow$  M: Mantisse, E exposant, et B base (2 ou 16)
- **Exemple:**
  - $-6.625 \Leftrightarrow$ 

32	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	0	0	0	0	1	1	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
  - $-6.625 \Leftrightarrow 6.625 \Leftrightarrow 110,1010 \Leftrightarrow 1,101010.2^2$
  - Sur 23 bits: 1,1010 1000 0000 0000 0000 000
  - Exposant sur 8 bits:  $E = \text{Décalage} + \text{base} = 2 + 127 = 129 \Leftrightarrow 10000001$

## C Types caractères

- Codé sur un octet
- Représenté par les codes ASCII 0-255
- Représentation d'un caractère entre apostrophe: 'a'
- Caractère d'échappement significatif : \n retour à la ligne, \t tabulation, ...
- Multiples représentations : 'A' ⇔ '\x41' ⇔ '\101'
- Extrait du tableau ascii:

Dec	Hex	Name	Char	Ctrl-char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	0	Null	NUL	CTRL-@	32	20	Space	64	40	@	96	60	`
1	1	Start of heading	SOH	CTRL-A	33	21	!	65	41	A	97	61	a
2	2	Start of text	STX	CTRL-B	34	22	"	66	42	B	98	62	b
3	3	End of text	ETX	CTRL-C	35	23	#	67	43	C	99	63	c
4	4	End of xmit	EOT	CTRL-D	36	24	\$	68	44	D	100	64	d
5	5	Enquiry	ENQ	CTRL-E	37	25	%	69	45	E	101	65	e
6	6	Acknowledge	ACK	CTRL-F	38	26	&	70	46	F	102	66	f
7	7	Bell	BEL	CTRL-G	39	27	'	71	47	G	103	67	g
8	8	Backspace	BS	CTRL-H	40	28	(	72	48	H	104	68	h

## C Initialisation de types et constantes

- Initialisation = déclaration et affectation
- **Exemple** : int n = 15 ;
- Constantes => « const »
  - const int p = 20 ;
  - Toute modification de la variable « p » est rejetée par le compilateur
  - Différent de #define => p ne peut intervenir dans les expressions constantes

## C Opérateurs arithmétiques

- Opérateurs classiques binaires à 2 opérandes : « + », « - », « \* », « / »
- Unaire sur une opérande : « - »
- Entre 2 opérandes de même type => cas échéant : conversion implicite gérée par le compilateur
- Opérateur modulo : %, reste de la division euclidienne  $11\%4 = 3$
- Remarque sur la division : « / »
  - En « int »:  $5/2 = 2$
  - En « float »:  $5.0/2.0$  vaut 2,5

## C Propriétés relatives des opérateurs

- Règle naturelle de priorité
- Opérateurs unaire prioritaire
- « \* » et « / » prioritaire à « + » et « - »
- Associativité de gauche à droite

## C Ce qu'on a couvert

---

- Notion de type
- Types entiers, flottants et caractère
- Opérateurs arithmétiques et leurs propriétés



**Alphorm**.com

Types, opérateurs  
et expressions

---

Affectation,  
incrémentation et  
ordre

Site : <http://www.alphorm.com>  
Blog : <http://blog.alphorm.com>

**Rabah ATTIK**

Formateur et Consultant indépendant  
Ingénierie système embarqué

## C Plan

---

- Affectation
- Incrémentation
- Ordre et opérateurs relationnels



## C Affectation

---

- Opérateurs d'affectation '=' affecter une valeur à une variable
  - Met en relation 2 opérandes
  - L'opérande de gauche doit pouvoir être affecté par l'opérande de droite
- Notion de 'lvalue' :
  - C'est une référence à un emplacement mémoire dont on pourra effectivement modifier la valeur
- Associativité de droite à gauche, peut entrainer une conversion
- Opérateurs d'affectation élargie :
  - lvalue = lvalue opérateur expression ⇔ lvalue opérateur = expression
  - '+=' '-=' '\*=' '/=' '%=' '|=' '^=' '&=' '<=' '>='



## C Incrémentation

- Opérateurs « ++ » et « -- »
- $i++ \Leftrightarrow i <- i+1$
- $i -= c \Leftrightarrow i <- i-c$
- Pre/post in(de)crémentation: ++i et i++ (i-- et --i)
  - ++lvalue => pre-incrémentation
  - lvalue++ => post-incrémentation
  - **Exemple:**

```
int n = 5;  
n = i++ - 5;  
=> n = 0
```

## C Ordre et opérateurs relationnels

- Les opérateurs relationnels :

Opérateur	Signification
<	Inférieur à
<=	Inférieur ou égal à
>	Supérieur à
>=	Supérieur ou égal à
==	Égal à
!=	Différent de

- Priorité: '<' '<=' '>=' et '>' sont de même priorité et supérieure à '&#x2192;' et '&#x2192;'
  - $a < b == c < d \Leftrightarrow (a < b) == (c < d)$
- Priorité inférieure aux opérateurs arithmétiques

## C Ce qu'on a couvert

---

- Affectation
- Incrémentation
- Ordre et opérateurs relationnels



**Alphorm**.com

Types, opérateurs  
et expressions

---

Opérateurs logiques

Site : <http://www.alphorm.com>  
Blog : <http://blog.alphorm.com>

**Rabah ATTIK**

Formateur et Consultant indépendant  
Ingénierie système embarqué

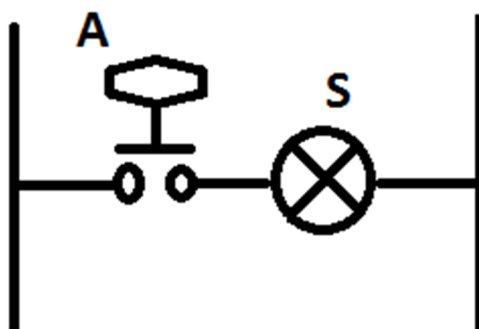
## C Plan

- Rappel sur la logique booléenne
  - « ET » logique (AND)
  - « OU » logique (OR)
  - « OU » exclusif (XOR)
  - Négation (NOT)
- Les opérateurs logiques
  - Opérateurs logiques booléens
  - Opérateurs logiques binaire ou « bit-à-bit »



## C Rappel sur la logique booléenne

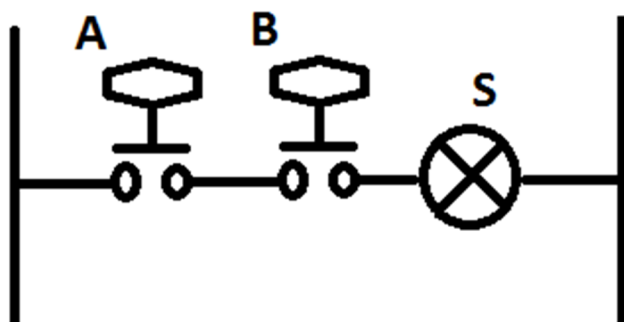
- Logique binaire à 2 états : 0 ou 1, vrai ou faux



A	S
0	0
1	1

## C Rappel sur la logique booléenne

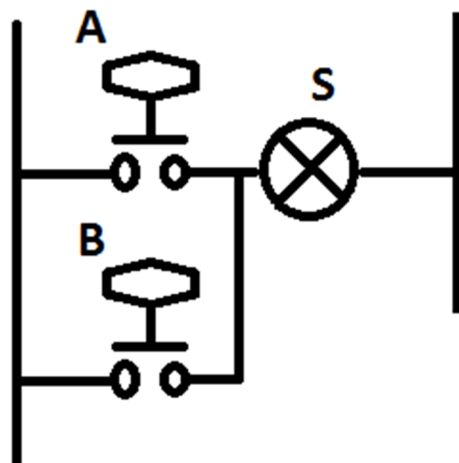
- ET Logique



A	B	S
0	0	0
0	1	0
1	0	0
1	1	1

## C Rappel sur la logique booléenne

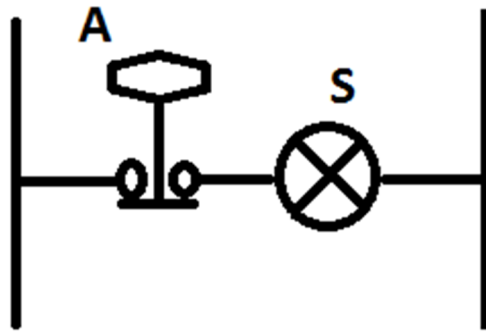
- OU logique



A	B	S
0	0	0
0	1	1
1	0	1
1	1	1

## C Rappel sur la logique booléenne

- NON logique



A	S
0	1
1	0

## C Les opérateurs logiques booléens

- ET: &&
- OU: ||
- Non: !
- Exemple:
  - `test = (a<b) || (c<d)`
  - `test2 = ! (a>b)`
  - `int s = a<10 && b >20`

## C Remarques

---

- Test d'expression logique et opérateur '&&'
- (a<b) && (c<d)
- if ( i<max && (c=getchar() != '\n') )

## C Les opérateurs logiques binaires

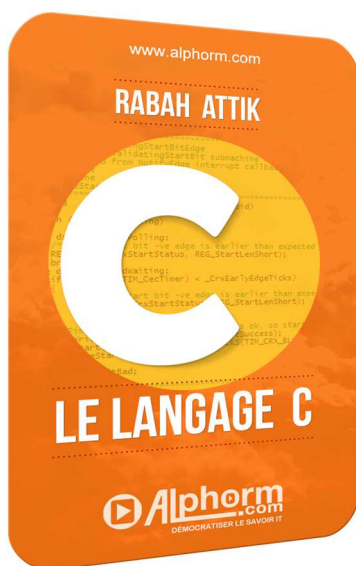
---

- ET: &
- OU: |
- OU exclusif: ^
- NON: ~
- Exemple:
  - int a=10, b=11; char test = a&b;
  - char test2 = ~a;

## C Ce qu'on a couvert

---

- Rappel sur la logique booléenne
- Opérateurs logiques



**Alphorm**.com

Types, opérateurs  
et expressions

---

Autres opérateurs  
et priorités

Site : <http://www.alphorm.com>  
Blog : <http://blog.alphorm.com>

**Rabah ATTIK**

Formateur et Consultant indépendant  
Ingénierie système embarqué

## C Plan

---

- Opérateur cast
- Opérateur conditionnel
- Opérateur séquentiel
- Opérateur *sizeof*
- Priorités des opérateurs



## C Opérateur cast

---

- Convertir une expression dans un type voulu
- Exemple: `int n,p;`  
`double d = (double) n/p;`



## C Opérateur conditionnel

- « ... ? ... : ... »  $\Leftrightarrow$  condition ? ExpressionA : ExpressionB
- Exemple : **If(a>b)**  
                  **max = a;**  
          **Else**  
                  **max = b;**
- **max = (a>b) ? a : b ;** se lit:
  - si **a** est supérieur max vaut **a** sinon max vaut **b**

## C Opérateur séquentiel

- « , » enchaîner des expressions ou des instructions
  - **i++ , j = i+k , j--**
- Utilisé dans la structure conditionnelle  
      **if( i++,k>0) ...** remplace **i++; if(k>0) ...**

## C Opérateur sizeof

- Taille en mémoire d'un type
  - sizeof(type)
- Exemple :
  - int n = 14; sizeof(n) vaut 4
  - double d = 120; sizeof(d) vaut 8

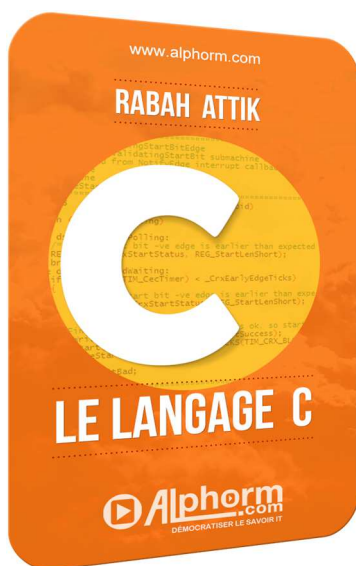
## C Priorités des opérateurs

Catégorie	Opérateurs	Associativité
Référence	() [] -> .	--->
unaire	+ - ++ -- ! ~* & (cast) sizeof	<---
arithmétique	* / %	--->
arithmétique	+ -	--->
décalage	<< >>	--->
relationnels	< <= > >=	--->
relationnels	== !=	--->
manipulation de bits	&	--->
manipulation de bits	^	--->
manipulation de bits		--->
logique	&&	--->
logique		--->
conditionnel	? :	--->
affectation	= += -= *= /= %= &= ^=  = <<= >>=	<---
sequentiel	,	--->

## C Ce qu'on a couvert

---

- Opérateur cast, conditionnel, séquentiel, *sizeof*
- Priorités de tous les opérateurs



**Alphorm**.com

Types, opérateurs  
et expressions

---

Déclaration  
des variables

Site : <http://www.alphorm.com>  
Blog : <http://blog.alphorm.com>

**Rabah ATTIK**

Formateur et Consultant indépendant  
Ingénierie système embarqué

## C Plan

---

- Déclaration de variable
- Les spécificateurs de type (short, long, signed, unsigned)
- Les types (int, char, float, double)
- Les classes de mémorisation (auto, static, extern, register)
- Les qualificateurs (const, volatile)
- Le déclarateurs



## C Déclaration de variable

---

- => Associer un spécificateur de type avec son type (éventuellement complété de qualificateur et d'une classe de mémorisation à un déclarateur

## C Spécificateurs de types

---

- short,
- Long,
- unsigned / signed

## C Types

---

- int
  - 2 ou 4 octets dépend du spécificateur
- char
  - 1 octet / Le domaine de valeur dépend du spécificateur
- float / double / long double
  - Pas de spécificateurs short unsigned ou signed

## C Classes de mémorisation

---

- automatic
- static
- extern
- register

## C Qualificateurs

---

- const
  - Non modification d'une variable pendant l'exécution
- volatile
  - Modification indépendamment des instructions du programme

## C Déclarateur

---

- Suit les règles imposées sur les identificateurs
  - Suite de caractères composés de
    - Lettre / chiffre et caractère « \_ »
  - Ne doit pas commencer par un chiffre
  - La casse est prise en compte
  - Exemple :
    - `_ma_variable`      `MaVariable`      `mavariabLe`
    - `9mavariabLe` => interdit

## C Déclaration de variable

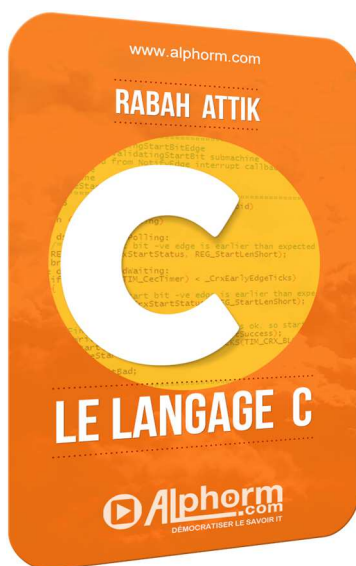
---

- `static const unsigned int _scui = 14;`
- `static double sld = 1E16;`
- `auto unsigned float auf = 12.35;`

## C Ce qu'on a couvert

---

- Déclaration de variables
- Les différents mots clefs d'une déclaration



**Alphorm**.com

## Structures de contrôle

---

## Notions de blocs

Site : <http://www.alphorm.com>  
Blog : <http://blog.alphorm.com>

**Rabah ATTIK**

Formateur et Consultant indépendant  
Ingénierie système embarqué



## C Plan

- Présentation des structures de contrôle
- Définition d'un bloc de code
- Les blocs de fonctions et le bloc « main »
- Déclarations dans un bloc et classe d'allocation



## C Présentation des structures de contrôle

Type d'instructions	Instructions	Caractéristiques
Instructions simples	break goto continue return	Branchement inconditionnel
Instructions structurées	Choix: if / switch Boucles: do...while while for	Contient au moins une instruction
Bloc	{ ... }	Regroupement de 0,1 ou plusieurs instructions quelconques

## C Définition d'un bloc de code

- Bloc = suite d'instructions exécutables placées entre accolades

```
{  
....  
.....  
}
```

- Instructions :
  - Déclarations (c90 vs >c99)
  - Instructions simples
  - Instructions structurées
  - Autres blocs

## C Bloc de fonctions et bloc main

- Fonction et main sont des blocs

- Fonction :

```
Type de retour fonction(argument,...)  
{  
    déclarations;  
    instructions;  
}
```

- main : l'entrée du programme
  - Au passage, les prototypes de main
    - int main (int argc, char \*\* argv)
    - int main(void)

## C Déclaration et classe d'allocation

- Exemple :

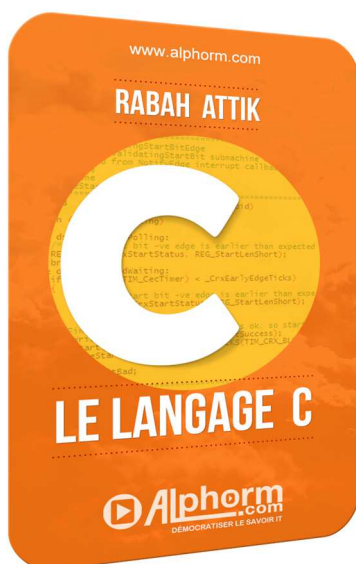
```
int f()  
{  
    int i,j;  
    {  
        int k  
        k=5;  
    }  
    k=6;  
    i=7;  
    j=k++;  
}
```

## C Déclaration et classe d'allocation

- Par défaut la classe d'allocation est : auto (automatique)
  - => Emplacement alloué à l'entrée du bloc et libéré en sortie
- Il existe d'autres classe d'allocation :
  - Statique / Automatique / registre / Externe

## C Ce qu'on a couvert

- Présentation des structures de contrôle
- Définition d'un bloc de code
- Les blocs de fonctions et le bloc « main »
- Déclarations dans un bloc et classe d'allocation



**Alphorm**.com

## Structures de contrôle Notions de blocs

Site : <http://www.alphorm.com>  
Blog : <http://blog.alphorm.com>

**Rabah ATTIK**

Formateur et Consultant indépendant  
Ingénierie système embarqué

## C Plan

- Présentation des structures de contrôle
- Définition d'un bloc de code
- Les blocs de fonctions et le bloc « main »
- Déclarations dans un bloc et classe d'allocation



## C Présentation des structures de contrôle

Type d'instructions	Instructions	Caractéristiques
Instructions simples	break goto continue return	Branchement inconditionnel
Instructions structurées	Choix: if / switch Boucles: do...while while for	Contient au moins une instruction
Bloc	{ ... }	Regroupement de 0,1 ou plusieurs instructions quelconques

## C Définition d'un bloc de code

- Bloc = suite d'instructions exécutables placées entre accolades

```
{  
....  
.....  
}
```

- Instructions :
  - Déclarations (c90 vs >c99)
  - Instructions simples
  - Instructions structurées
  - Autres blocs

## C Bloc de fonctions et bloc main

- Fonction et main sont des blocs

- Fonction :

```
Type de retour fonction(argument,...)  
{  
    déclarations;  
    instructions;  
}
```

- main : l'entrée du programme
  - Au passage, les prototypes de main
    - int main (int argc, char \*\* argv)
    - int main(void)

## C Déclaration et classe d'allocation

- Exemple :

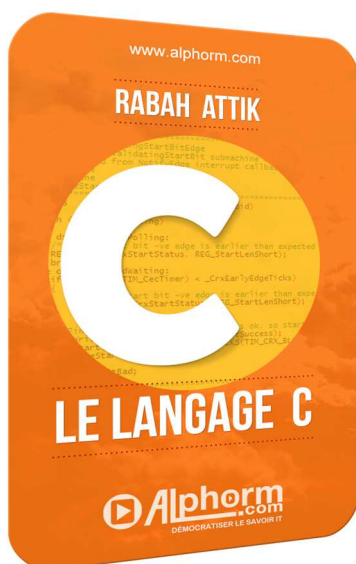
```
int f()  
{  
    int i,j;  
    {  
        int k  
        k=5;  
    }  
    k=6;  
    i=7;  
    j=k++;  
}
```

## C Déclaration et classe d'allocation

- Par défaut la classe d'allocation est : auto (automatique)
  - => Emplacement alloué à l'entrée du bloc et libéré en sortie
- Il existe d'autres classe d'allocation :
  - Statique / Automatique / registre / Externe

## C Ce qu'on a couvert

- Présentation des structures de contrôle
- Définition d'un bloc de code
- Les blocs de fonctions et le bloc « main »
- Déclarations dans un bloc et classe d'allocation



**Alphorm**.com

### Structures de contrôle

Structures de choix : « **if** »,  
« **switch** »

Site : <http://www.formation.inge-tech.com>  
Blog : <http://blog.alphorm.com>

Rabah ATTIK

Formateur et Consultant indépendant  
Ingénierie système embarqué



## C Plan

---

- Structures de choix
  - Syntaxe et rôle de l'instruction « **if** »
  - Cas des « **if** » imbriqués
  - Syntaxe et rôle de l'instruction « **switch** »
  - « **if** » vs « **switch** »



## C Syntaxe et rôle de l'instruction if

---

- **if (expression) instruction\_1 else instruction\_2**
- **if (expression) instruction**
- Permet de programmer une structure dite de choix (ou sélection ou alternative)

## C Cas des « if » imbriqués

- Les instructions des if sont quelconques et peuvent renfermer d'autres instructions « if »

```
if (expression_1) if(expression_2) instruction_1  
                else instruction_2
```

```
if (expression_1) instruction_1  
                else if(expression_2) instruction_2  
                    if(expression_3) instruction_3  
                    ...
```

## C Syntaxe et rôle de l'instruction switch

- Syntaxe :

```
switch (expression_1) {  
    case constante_1: suite_1 d'instructions;  
    case constante_2: suite_2 d'instructions;  
    case constante_3: suite_3 d'instructions;  
    ...  
    default: suite_n d'instructions;  
}
```

- Rôle :

- Programme une structure de choix multiples

## C « if » vs « switch »

---

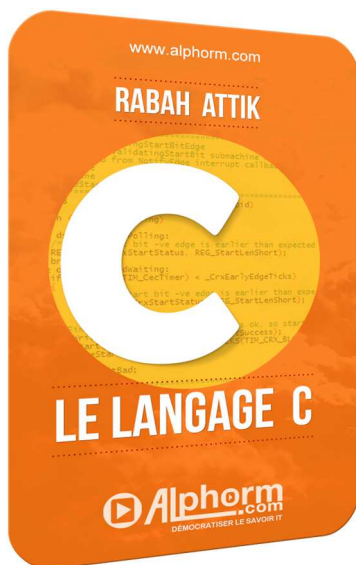
- Limitation du switch :
  - Condition porte sur des valeurs entières
  - L'énumération trop longue devient illisible
- Énumération courte et condition entière => switch
- Condition non entière => if

## C Ce qu'on a couvert

---

- Syntaxe et rôle des structures de choix
- « if » vs « switch »





**Alphorm**.com

## Structures de contrôle

---

### Contrôle de boucle, return et goto

Site : <http://www.alphorm.com>  
Blog : <http://blog.alphorm.com>

Rabah ATTIK

Formateur et Consultant indépendant  
Ingénierie système embarqué

Formation Le langage C

alphorm.com™©

## C Plan

---

- Contrôle de boucle
  - Instruction « **break** »
  - Instruction « **continue** »
- Instruction « **return** »
- Instruction « **goto** »



Formation Le langage C

alphorm.com™©

## C Contrôle de boucle

---

- Instructions de branchement inconditionnel = contrôle de boucle
- « break », « continue »
- Usage dans les boucles et la structure de choix switch => « break » et « continue »

## C Instruction « break »

---

- break;
  - Sortir d'une boucle
    - Boucle à sortie intermédiaire
    - Boucle à sorties multiples
  - Mettre fin à l'exécution de l'instruction dans un **switch**
  - Ne concerne pas l'instruction « if »
  - Structures imbriquées : break ne fait sortir que du niveau le plus interne

## C Instruction « break »

- break; Boucle à sortie intermédiaire

```
int i;  
for(i=1; i<=10 ; i++)  
{  
    if(2*i == 8);  
    break;  
}
```

```
while(1)  
{  
    instruction_1;  
    if(condition) break;  
    instruction_2;  
}
```

OU

```
bool sortie = 1  
do  
{ instruction_1;  
    if(condition) sortie =0;  
    else instruction_2;  
}while(sortie)
```

## C Instruction « break »

- break; Boucle à sorties multiples

```
while(1)  
{  
    instruction_1;  
    if(condition_1) break;  
    instruction_2;  
    if(condition_2) break;  
    instruction_3;  
    if(condition_3) break;  
}
```

OU

```
bool sortie = 1  
do  
{ instruction_1;  
    if(condition_1) sortie =0;  
    if(!sortie) { instruction_2}  
    if(condition_2) sortie =0;  
    if(!sortie) { instruction_3}  
    if(condition_3) sortie =0;  
    if(!sortie) { instruction_4}  
}while(sortie)
```

## C Instruction « break »

- break; cas du switch

```
switch (expression_1) {  
    case constante_1: suite_1 d'instructions;  
    case constante_2: suite_2 d'instructions;  
    break;  
    case constante_3: suite_3 d'instructions;  
    break;  
    ...  
    default: suite_n d'instructions;  
}
```

## C Instruction « continue »

- « continue » permet de forcer le passage au tour suivant dans la boucle
- Les instructions entre le **continue** et la fin de la boucle sont ignorées

```
int n;  
do  
{  
    printf(« Veuillez entrer un nombre positif »);  
    scanf(« %d », &n);  
    if (n < 0)  
    {  
        printf(« Le nombre est négatif »);  
        continue;  
    }  
    printf(« Le carré de %d est %d\n », n, n*n);  
}  
while(n)
```

- Structure imbriquées : **continue** ne concerne que les boucles les plus internes

## C Instruction « return »

---

- return;
  - Sortir de fonction donc de toutes boucles ou structure de contrôle
  - Utilisé pour retourner un type de retour dans un fonction

## C Instruction « goto »

---

- goto label;
  - Instruction de saut permettant de poursuivre l'exécution du programme en un autre point
  - label = étiquette
  - Saut ?
    - Pas de sauts en dehors d'une fonction.
    - Possible en dehors et à l'intérieur des blocs d'instructions sous certaines conditions.
    - Emplacement du label: Si la destination du saut se trouve après une déclaration, cette déclaration ne doit pas comporter d'initialisations + déclaration d'un type simple



## C Instruction « goto »

---

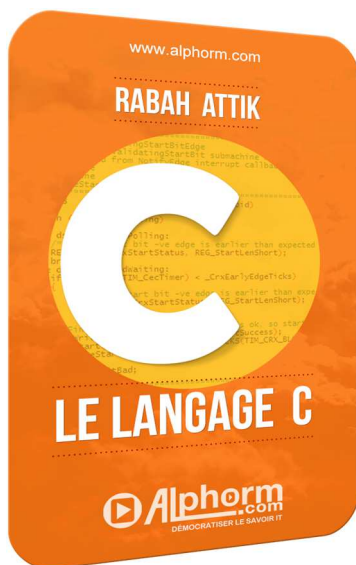
- goto label;
  - Utilité ?
    - Sortie de structures imbriquées
    - Gestion des erreurs
    - Usage critiqué car générateur d'erreurs et peut être remplacé la plupart du temps par « break » et « continue »

## C Ce qu'on a couvert

---

- Les instructions de contrôle de boucle





**Alphorm**.com

## Structures de contrôle

---

# Introduction à l'algorithmique

Site : <http://www.alphorm.com>  
Blog : <http://blog.alphorm.com>

Rabah ATTIK

Formateur et Consultant indépendant  
Ingénierie système embarqué

Formation Le langage C

alphorm.com™©

## C Plan

---

- Algorithmique : définition
- Algorithmique : caractéristiques
- Algorithmique : représentation
- Pseudo-code et programmation
- Les algorithmes de tri quadratique
  - Tri par insertion (détaillé)
  - Tri par sélection
  - Tri à bulles



Formation Le langage C

alphorm.com™©

## C Algorithmique : définition

- Un algorithme est la description univoque d'une méthode effective pour résoudre un problème, exprimée à l'aide d'une suite d'instructions élémentaires
- Un algorithme est une suite ordonnée d'instructions qui indique la démarche à suivre pour résoudre une série de problèmes équivalents.
- **Wikipedia** : « *Le mot algorithme vient du nom latinisé du mathématicien perse [Al-Khawarizmi](#), écrivant en langue [arabe](#), surnommé « le père de l'algèbre »<sup>2</sup>. Le domaine qui étudie les algorithmes est appelé [l'algorithmique](#).* »

## C Algorithmique : caractéristiques

- **Robustesse** : aptitude à se protéger des conditions anormales d'utilisation
- **Réutilisabilité** : aptitude à être réutilisé pour résoudre des tâches équivalentes (à celle pour laquelle il a été conçu)
- **Complexité** : Nombre d'instructions élémentaires à exécuter
- **Efficacité** : Aptitude à utiliser de manière optimale les ressources matériel qui l'exécute

Exemple

Recherche un mot dans un dictionnaire  
Recette de cuisine

Robuste ?  
Réutilisable ?  
Complexe ?  
Efficace ?

## C Algorithmique : caractéristiques

- **Robustesse** : aptitude à se protéger des conditions anormales d'utilisation
- **Réutilisabilité** : aptitude à être réutilisé pour résoudre des tâches équivalentes (à celle pour laquelle il a été conçu)
- **Complexité** : Nombre d'instructions élémentaires à exécuter
- **Efficacité** : Aptitude à utiliser de manière optimale les ressources matériel qui l'exécute

Exemple

Recherche un mot dans un dictionnaire

Robuste ?  
Réutilisable ?  
**Complexe ?**  
Efficace ?

## C Algorithmique : caractéristiques

- **Complexité** : Mesure de la complexité en  $O(n)$ 
  - Estimation du nombre d'instructions élémentaires à exécuter  $Nb(n)$

**Si il existe  $k$  /  $Nb(n) < k*n$  Alors on dit que la complexité est en  $O(n)$**

**Si il existe  $k$  /  $Nb(n) < k*n^2$  Alors on dit que la complexité est en  $O(n^2)$**

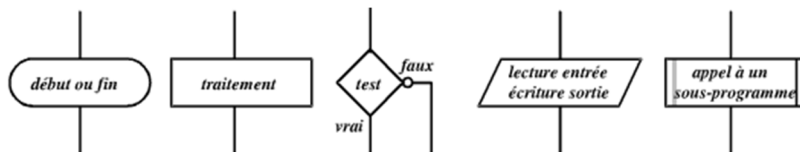
**Si il existe  $k$  /  $Nb(n) < k*\log(n)$  Alors on dit que la complexité est en  $O(\log(n))$**

...

- D'autres notions de complexité:
  - Complexité en temps et en mémoire

## C Algorithmique : représentation

- Organigramme

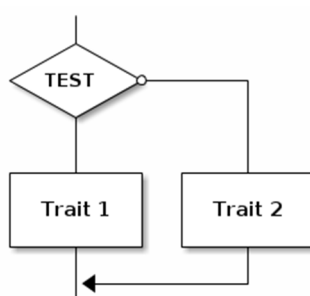


- Pseudo-code

- Proche du code, conventionnel, pas de syntaxe,
- Non reconnu par la machine

## C Algorithmique : représentation

Organigramme



Pseudo-code

```
Si TEST, alors
| Traitement 1
Sinon
| Traitement 2
Fin Si
```

## C Pseudo-code et programmation

### Pseudo-code

```
Algorithme AfficheTableau (t : tableau)
{ Affiche tous les éléments d'un tableau }
Variable i : entier
Début
  Pour i ← 1 à taille(t) faire
    Écrire(t[i])
  Fin Pour
Fin
```

### Programmation

```
/*Code C*/
int AfficheTableau (int t[])
{
  int i;
  for(i=0;i<N;i++)
  {
    printf(« valeur de t[%d]:%d »,i,t[i]);
  }
}
```

## C Les algorithmes de tri quadratiques

- Qu'est ce qu'un tri ?
  - Tri dans un ordre des éléments d'un tableau ou d'une liste



- 3 algorithmes de tri quadratique:
  - Tri par sélection
  - Tri par insertion
  - Tri par bulles

## C Les algorithmes de tri quadratiques

- Tri par sélection: Principe
  - On sélectionne le plus grand élément
  - On le place en dernière position par un échange
  - On réitère le procédé sur les n-1 éléments restants

## C Les algorithmes de tri quadratiques

- Tri par sélection: Exemple

6	3	7	2	3	5
6	3	5	2	3	7
6	3	5	2	3	7
3	3	5	2	6	7
3	3	5	2	6	7
3	3	2	5	6	7
3	3	2	5	6	7
2	3	3	5	6	7
2	3	3	5	6	7

## C Les algorithmes de tri quadratiques

- Tri par sélection: pseudo-code

```
PROCEDURE TriSelection(entier *T, entier n)
début
    entier k, i, imax, temp;
    pour k ← n-1 à 1 pas -1 faire
        /* recherche de l'indice du maximum : */
        imax ← 0;
        pour i ← 1 à k pas 1 faire
            si T[imax] < T[i] faire
                imax ← i;
            fin faire
        fin faire
        /* échange : */
        temp ← T[k];
        T[k] ← T[imax];
        T[imax] ← temp;
    fin faire
fin
```

## C Les algorithmes de tri quadratiques

- Tri par sélection: complexité
  - Recherche du maximum  $\sim (n-1) + (n-2) + \dots + 2 + 1 = n*(n-1) / 2$
  - Echange =  $3 * (n-1) \sim (n-1)$
  - Total  $\sim (n-1) + n*(n-1)/2$
- Complexité en  $O(n^2)$



## C Les algorithmes de tri quadratiques

- Tri par insertion: Principe
  - On tri les 2 premiers éléments
  - On insère le 3eme à sa place pour avoir une liste triée de 3 éléments
  - On insère le 4eme à sa place pour avoir une liste triée de 4 éléments
  - ...
- Complexité en  $O(n^2)$

## C Les algorithmes de tri quadratiques

- Tri par bulles: Principe
  - On échange 2 éléments successifs s'ils ne sont pas dans l'ordre
    - On fait remonter le maximum à sa place
- Complexité en  $O(n^2)$

## C Ce qu'on a couvert

- Algorithmique: définition, caractéristiques, représentation
- Pseudo-code et programmation
- Exemples avec les tris quadratiques
  - Par sélection, par insertion et par bulles



**Alphorm**.com

Tableaux, pointeurs et chaines

Les pointeurs

Site : <http://formation.inge-tech.com>  
Blog : <http://www.alphorm.com/blog>  
Forum: <http://www.alphorm.com/forum>

**Rabah ATTIK**

Formateur et consultant indépendant  
Ingénierie système embarqué  
[rabah.attik@inge-tech.com](mailto:rabah.attik@inge-tech.com)

## C Plan

---

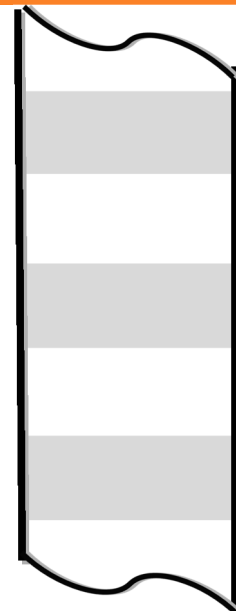
- Les pointeurs
  - La mémoire, une question d'adresse !
    - Rappel sur la mémoire, et les types
- Définition et utilisation des pointeurs
- Introduction à l'arithmétique des pointeurs
- Pointeur générique « **void\*** » et pointeur NULL



## C La mémoire, une question d'adresse

---

- L'organisation de la mémoire
  - Mémoire centrale = mémoire vive
  - Constituée d'octets identifiés par une adresse
  - Permet de stocker les données d'un programme
  - Connectée à l'unité centrale via 2 bus
    - Bus d'adresse
    - Bus de données



## C La mémoire, une question d'adresse

- Rappel sur les types et les variables
  - Les variables sont déclarées avec un type et un identificateur
  - Ce sont des « lvalue »

```
int i , j;
i = 5;
j = 78;
```

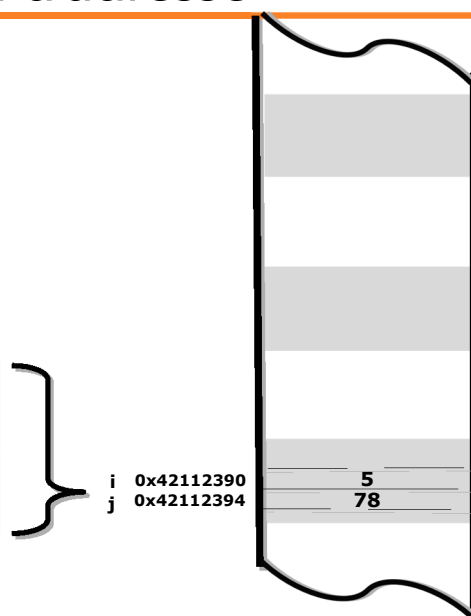
Objet	Adresse	Valeur
i	0x42112390	5
j	0x42112394	78

## C La mémoire, une question d'adresse

- Petit rappel sur les types et les variables

```
int i , j;
i = 5;
j = 78;
```

Objet	Adresse	Valeur
<b>i</b>	<b>0x42112390</b>	<b>5</b>
<b>j</b>	<b>0x42112394</b>	<b>78</b>



## C La mémoire, une question d'adresse

- Les différents types de données

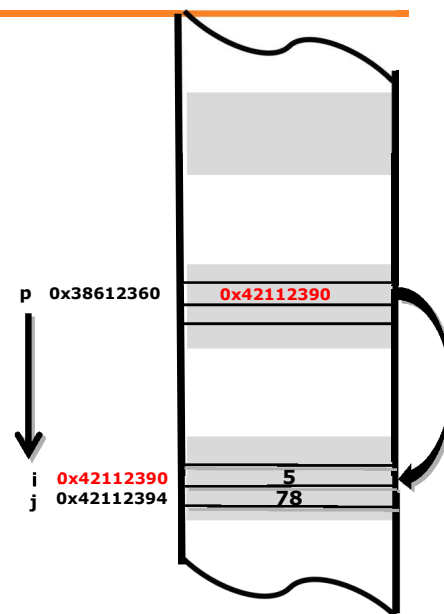
Type de donnée	Signification	Taille (en octets)	Plage de valeurs acceptée
char	Caractère	1	-128 à 127
unsigned char	Caractère non signé	1	0 à 255
short int	Entier court	2	-32 768 à 32 767
unsigned short int	Entier court non signé	2	0 à 65 535
int	Entier	2 (sur processeur 16 bits) 4 (sur processeur 32 bits)	-32 768 à 32 767 -2 147 483 648 à 2 147 483 647
unsigned int	Entier non signé	2 (sur processeur 16 bits) 4 (sur processeur 32 bits)	0 à 65 535 0 à 4 294 967 295
long int	Entier long	4	-2 147 483 648 à 2 147 483 647
unsigned long int	Entier long non signé	4	0 à 4 294 967 295
float	Flottant (réel)	4	$3.4 \cdot 10^{-38}$ à $3.4 \cdot 10^{38}$
double	Flottant double	8	$1.7 \cdot 10^{-308}$ à $1.7 \cdot 10^{308}$
long double	Flottant double long	10	$3.4 \cdot 10^{-4932}$ à $3.4 \cdot 10^{4932}$

Formation Le langage C

alphorm.com™©

## C Définition et utilisation des pointeurs

- Un pointeur est une variable contenant l'adresse d'une autre variable d'un type donné.
- Alternative à la manipulation d'adresse
  - Une adresse n'est pas une lvalue
- Sert à manipuler des structures de données
- Utilisée pour la passage par adresse en arguments d'une fonction
  - Les variables passées par valeur ne vivent que sur la portée du bloc



Formation Le langage C

alphorm.com™©

## C Définition et utilisation des pointeurs

- Déclaration d'un pointeur sur une variable de type « int »
  - `int * p_int;`
- Initialisation d'un pointeur
  - `p_int = NULL;`
- Affectation d'une adresse à un pointeur
  - `int i = 5;`
  - `int * p_int = &i ;`
- Déférencement
  - `*p_int = 5` -> Modifie la valeur à l'adresse pointée par `p_int`

## C Définition et utilisation des pointeurs

- Cas plus complet
  - `const int* p`
  - `const int * const p;`
  - `static const float * ptr_f;`
  - `static const float f = 12.36;`
  - `ptr_f = &f;`

## C Arithmétique des pointeurs

---

- Un pointeur contient l'adresse d'une **variable typée**
  - Opération arithmétiques entre pointeurs (Différence convertie vers le type ptrdiff\_t)
  - Incrémentation / décrémentation
  - Le déplacement dépend de la taille du type en mémoire

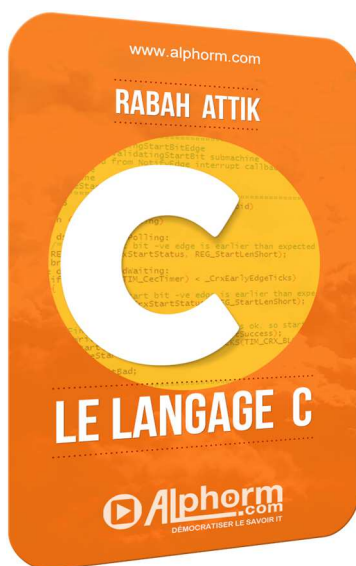
## C Pointeur générique et pointeur NULL

---

- Un pointeur générique est de type (void\*)
  - Cas d'opérations sur les adresses
  - Ne peut pas être déréférencé ou utilisé avec des opérations arithmétiques
- Le pointeur NULL
  - Ne pas confondre avec (pointe sur RIEN)
  - Valeur définie dans les fichiers stdlib.h / stddef.h / ...
  - Permet notamment une initialisation ou un test sur un pointeur

## C Ce qu'on a couvert

- Rappel sur la mémoire et les types
- Définition d'un pointeur
  - Utilisation d'un pointeur / cas du « const »
- Introduction à l'arithmétique des pointeurs
- Cas des pointeurs génériques et du pointeur NULL



**Alphorm**.com

Tableaux, pointeurs et chaines

Les tableaux

Site : <http://www.alphorm.com>  
Blog : <http://blog.alphorm.com>

**Rabah ATTIK**

Formateur et Consultant indépendant  
Ingénierie système embarqué



## C Plan

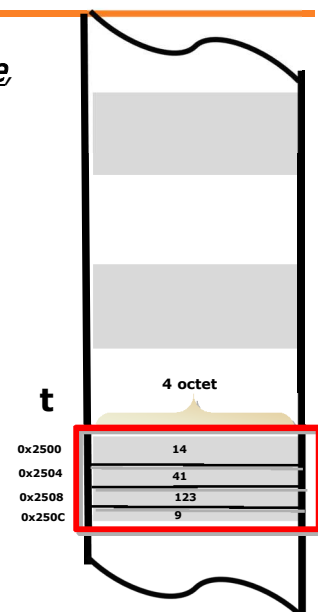
- Les tableaux
  - Précision sur les tableaux
    - Allocation en mémoire
    - Opérateur sizeof
- Les tableaux multidimensionnels



## C Tableaux

- *Un tableau est une suite de variables de même type, situées dans un espace contigu en mémoire*
- Déclaration d'un tableau:
  - `int t[4];`
- Initiation d'un tableau (indice à 0)
  - `t[0] = 14;`
  - `t[1] = 41;`
  - `t[2] = 123;`
  - `t[3] = 9;`

**OU** `int t[4] = {14, 41, 123, 9};`



## C Tableaux

---

- La déclaration permet de préciser:
  - Le nom donné au tableau
  - Le type de ses éléments
  - Eventuellement un ou deux qualificatifs (const, volatile)
  - Le nombre de ses éléments
  - Eventuellement une classe de mémorisation
    - Exemple: `static const unsigned int *ad, x, t[10];`

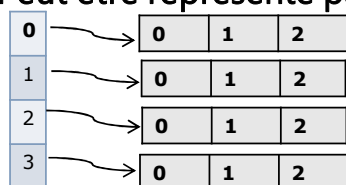
## C Tableaux

---

- `const int t[10];`
  - Un tableau de 10 entier constant
- `const int* tab1[5];`
  - `tab1` est un tableau de 5 pointeurs vers entier constant
- `sizeof` donne la taille du tableau en octet

## C Tableaux multidimensionnels: Cas 2D

- `int tab2D[4][3];`
  - `tab2D[4][3]` est un `int`
  - `tab2D[4]` est un tableau de 3 `int` (au sens de l'indice)
  - `tab2D` est un tableau à 5 « cases » dont chaque « case » contient un tableau de 3 `int`
- Peut être représenté par:

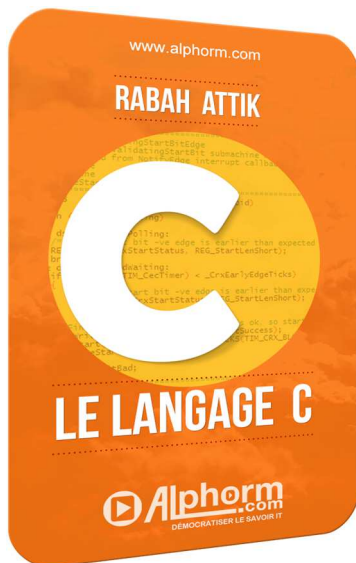


[0][0]	[0][1]	[0][2]	[1][0]	[1][1]	[1][2]	[2][0]	[2][1]	[2][2]	[3][0]	[3][1]	[3][2]
--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------

## C Ce qu'on a couvert

- Les tableaux
  - Précision sur les tableaux
- Les tableaux multidimensionnels
  - Cas 2D





**Alphorm**.com

Tableaux, pointeurs et chaines

## Liens entre pointeurs et tableaux

Site : <http://www.alphorm.com>  
Blog : <http://blog.alphorm.com>

Rabah ATTIK

Formateur et Consultant indépendant  
Ingénierie système embarqué

Formation Le langage C

alphorm.com™©

### C Plan

- Les pointeurs
- Les tableaux
- Lien entre pointeurs et tableaux



Formation Le langage C

alphorm.com™©

## C Les pointeurs

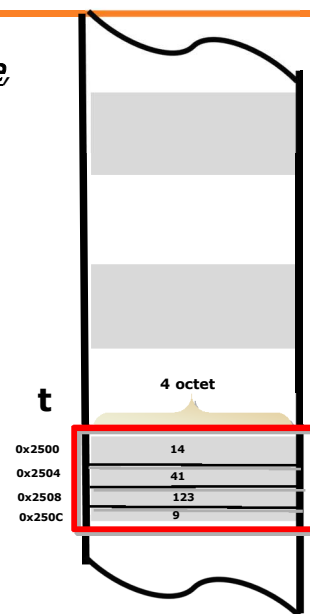
- Déclaration d'un pointeur sur une variable de type « int »
  - `int * p_int;`
- Initialisation d'un pointeur
  - `p_int = NULL;`
- Affectation d'une adresse à un pointeur
  - `int i = 5;`
  - `int * p_int = &i ;`
- Déférencement
  - `*p_int = 5` modifie la variable à l'adresse pointée par `p_int`

## C Tableaux

- *Un tableau est une suite de variables de même type, situées dans un espace contigu en mémoire*
- Déclaration d'un tableau:
  - `int t[4];`
- Initiation d'un tableau (indice à 0)
  - `t[0] = 14;`
  - `t[1] = 41;`
  - `t[2] = 123;`
  - `t[3] = 9;`

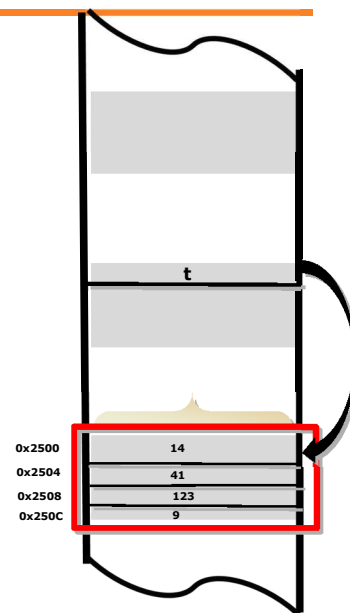
OU

`t = {14, 41, 123, 9};`



## C Lien entre pointeurs et tableaux

- Un tableau est un pointeur sur le premier élément
  - Déclaration d'un tableau: `int t[4];` t pointe vers `t[0]`
- Arithmétique des pointeurs
  - Addition, soustraction, incrémentation, décrémentation
  - $*(t+n) \iff t[n]$  (attention aux débordements)
  - Ex: Si `p1 = t` alors `p1 + 3` pointe sur `t[3]`
  - `<`, `>`, `<=`, `>=`, `==`, `!=`. La comparaison de deux pointeurs qui pointent *dans le même tableau* est équivalente à la comparaison des indices correspondants
  - Cas pratique



Formation Le langage C

alphorm.com™©

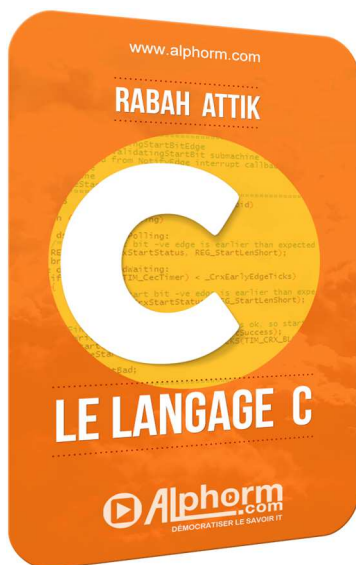
## C Ce qu'on a couvert

- Rappels sur les pointeurs et les tableaux
- Lien entre pointeurs et tableaux



Formation Le langage C

alphorm.com™©



**Alphorm**.com

## Tableaux, pointeurs et chaînes Chaînes de caractères

Site : <http://formation.inge-tech.com>  
Blog : <http://www.alphorm.com/blog>  
Forum : <http://www.alphorm.com/forum>

**Rabah ATTIK**

Formateur et consultant indépendant  
Ingénierie système embarqué  
rabah.attik@inge-tech.com

Formation Le langage C

alphorm.com™©

### **C** Plan

- Rappel sur le type caractère
- Chaîne de caractère et propriétés des constantes chaînes
- Manipuler les chaînes
- Entrées-sorties standards de chaînes
- Représentation ASCII des caractères



Formation Le langage C

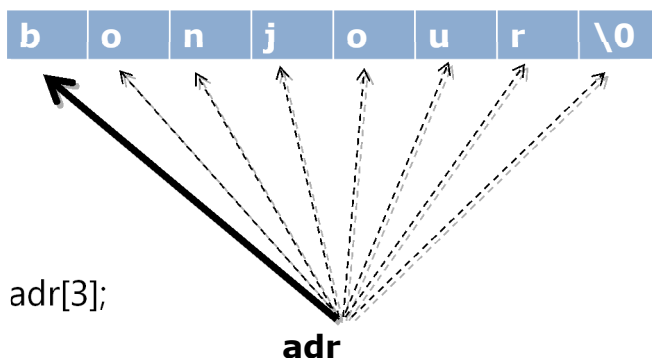
alphorm.com™©

## C Rappel sur le type char

- Type caractère: char
- Un caractère est déclaré avec '
- Codé sur 1 octet
- Exemples:
  - `char car = 'p';`
  - `const char p = 'g';`

## C Propriétés des constantes chaines

- Une chaine est un emplacement mémoire contiguë
- Suffixée par `\0` (de manière conventionnelle)
- `char * adr = « bonjour »;` correspond à :



- `char car = adr[3];`



## C Propriétés des constantes chaines

- Constantes chaines de classe d'allocation statique : l'emplacement mémoire est toujours le même pendant l'exécution du programme
- La modification de la chaîne est à priori possible (  $*(adr+3) = 'x'$  donne « bonxour », pour s'en protéger on utilise « const »
  - `char* adr = «bonjour»` devient `const char * adr = «bonjour»`

## C Manipuler les chaînes

- Créer utiliser ou modifier une chaîne
- Créer un tableau de chaîne
  - Utilisation d'un pointeur sur une zone mémoire existante
  - Allocation d'une zone mémoire par déclaration d'un tableau ou par allocation dynamique

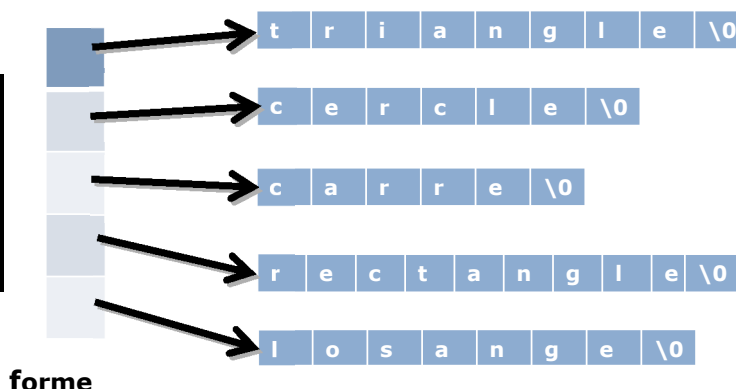
## C Manipuler les chaînes

- Créer utiliser ou modifier une chaîne
  - **Créer** : Déclaration sous forme de tableau / allouer avec malloc
    - `char* chaine = «bonjour»`
    - `char chaine[100] = «bonjour»;`
    - `char* chaine = (char*)malloc(100 * sizeof(char) );` puis utilisation du pointeur identifié par l'identificateur « chaine »
  - **Modifier**:
    - Avec les fonctions de la bibliothèque standard (`strcpy`)
    - Caractères par caractères

## C Manipuler les chaînes

- Créer un tableau de chaîne

```
char *forme[5];  
forme[0] = «triangle»;  
forme[1] = «cercle»;  
forme[2] = «carre»;  
forme[3] = «rectangle»;  
forme[4] = «losange»;
```



- Sécuriser en écriture le tableau:

```
char *forme[5]; => const char* forme[5]; => const char* const forme[5];
```

## C Entrées-sorties standards de chaînes

- Écriture de chaîne
  - puts
  - printf / fprintf
- Lecture de chaîne:
  - gets
  - scanf
- Les formats

## C puts / fputs

- int puts (const char \*chaîne)
  - int fputs (const char \*s, FILE \*stream);
- } (stdio.h)
- Valeur de retour non négative si Ok sinon EOF
  - Suivi de \n
  - fputs renvoi la chaîne dans un fichier texte quelconque

```
char ch1[] = «bonjour»;  
puts(ch1);
```

## C printf / fprintf

- Affichage sans saut de ligne « \n » et possibilité de formatage

```
int i;  
...  
float r = racine(i);  
  
printf("La racine carrée de %d est %.4f\n", i, r);
```

- Printf vs puts

```
char ch1[] = "bonjour";  
puts(ch1);
```

*équivalent à*

```
printf("« %s\n », ch1);
```

## C gets / fgets

- char \*gets(char \*chaine) (stdio.h)
  - Lire des chaînes se présentant sur des lignes différentes

## C scanf / fscanf

- Saisie d'entrée formatée de l'entrée standard (stdin) ou d'un fichier pour fscanf
- scanf(«%s », ch1);

## C Table ASCII

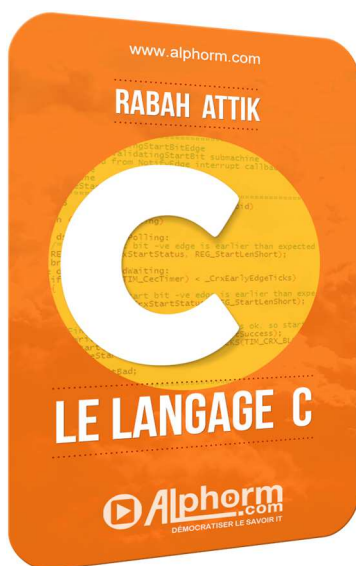
- <http://www.asciitable.com/>

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040	&#32;	Space	64	40	100	&#64;	@	96	60	140	&#96;	`
1	1	001	SOH (start of heading)	33	21	041	&#33;	!	65	41	101	&#65;	A	97	61	141	&#97;	a
2	2	002	STX (start of text)	34	22	042	&#34;	"	66	42	102	&#66;	B	98	62	142	&#98;	b
3	3	003	ETX (end of text)	35	23	043	&#35;	#	67	43	103	&#67;	C	99	63	143	&#99;	c
4	4	004	EOF (end of transmission)	36	24	044	&#36;	&	68	44	104	&#68;	D	100	64	144	&#100;	d

- Un code ASCII correspond à un caractère:
  - 256 code
  - Décimal / Héxadécimal / Octal / Html

## C Ce qu'on a couvert

- Rappel sur le type caractère
- Chaîne de caractère et propriétés des constantes chaînes
- Manipuler les chaînes
- Entrées-sorties standards de chaînes
- Table ASCII



**Alphorm**.com

## Les types structurés

### Structures et unions

Site : <http://formation.inge-tech.com>  
Blog : <http://www.alphorm.com/blog>  
Forum : <http://www.alphorm.com/forum>

**Rabah ATTIK**

Formateur et consultant indépendant  
Ingénierie système embarqué  
rabah.attik@inge-tech.com

## C Plan

---

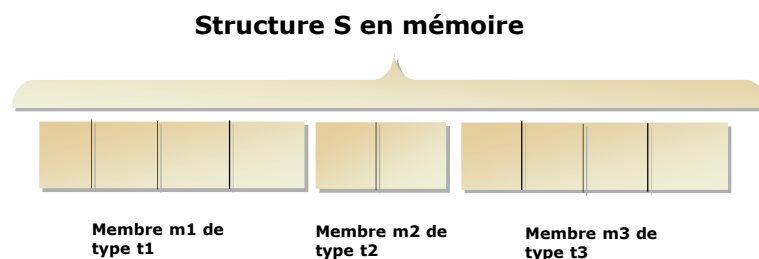
- Structure
  - Définition,
  - Déclaration et utilisation
  - Exemples
- Unions
  - Définition
  - Déclaration et utilisation
  - Exemples



## C Structure : Définition

---

- Type structuré, type agrégé
- Constitué de la réunion de plusieurs valeurs,
- Ces valeurs peuvent être de type différents



## C Structure: déclaration et utilisation

- Définition

```

1 struct point
2 {
3     char nom;      /* Nom du point      */
4     float x;       /* Abscisse        */
5     float y;       /* Ordonnée        */
6 }
    
```

- Utilisation

```

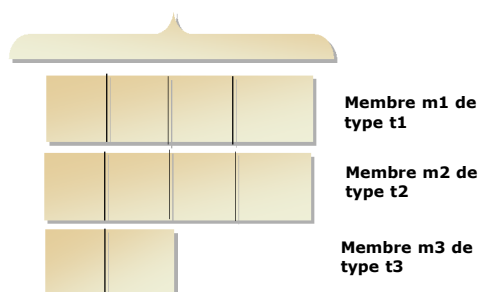
1 struct point p;
2 struct point ligne[NbrPoint];
    
```

- Pour accéder aux éléments on utilise les opérateurs . et ->
  - struct point p; p.x = 2;
  - struct point\* p\_s = Allocation ou récupération de pointeur puis p\_s->x = 2;

## C Union : définition

- Désigne un ensemble de variables de types différents susceptibles d'occuper alternativement une même zone mémoire
- La taille de la structure est celle du membre de type le plus grand

### Union U en mémoire





## C Union: déclaration et utilisation

- Définition

```
1 Union nombre
2 {
3     int i;
4     float x;
5     long l;
6 }
```

- Utilisation

```
1 union nombre n;
2 n.x = 12,5;
3 int i = n.i;
```

- Pour accéder aux éléments on utilise les opérateurs . et ->

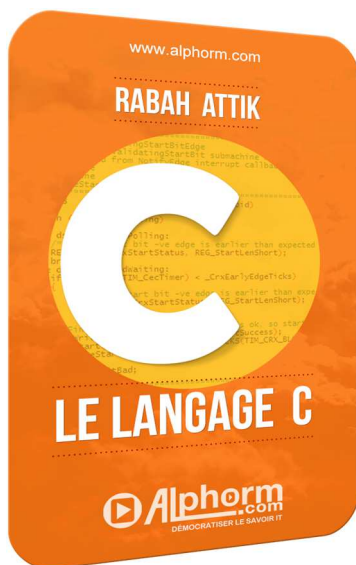
- union nombre u; u.x = 2;
- union nombre\* u\_s = Allocation ou récupération de pointeur puis u\_s->x = 2;

## C Ce qu'on a couvert

- Structure et unions

- Définition et usage
- Exemples





**Alphorm**.com

## Les types structurés typedef, les champs de bits, énumération

Site : <http://formation.inge-tech.com>  
Blog : <http://www.alphorm.com/blog>  
Forum : <http://www.alphorm.com/forum>

**Rabah ATTIK**

Formateur et consultant indépendant  
Ingénierie système embarqué  
rabah.attik@inge-tech.com

Formation Le langage C

alphorm.com™©

### C Plan

- Nouveau type ou synonyme avec « typedef »
- Champ de bit
- Énumération



Formation Le langage C

alphorm.com™©

## C Nouveau type avec « typedef »

- **Problématique**

- Déclarateur et spécificateur de type ne donnent pas toujours une information claire
  - *int v[3]* => v[3] est de type int et v est un tableau de 3 *int*
  - Cependant il n'est pas possible d'écrire *int[3] v*

## C Nouveau type avec « typedef »

- **Solution**

- Augmenter la clarté en définissant un type approprié avec *typedef*
- Exemples:

```
1  typedef int * ptr_int;  
2  ptr_int    p1,p2;  
3  typedef int vect[3];  
4  vect       v1,v2;  
5
```

- Syntaxe: *typedef specificateur\_de\_type [qualifieurs] liste\_de\_declarateurs;*
- *typedef* n'est pas un nouveau type mais un synonyme

## C Nouveau type avec « typedef »

- Un synonyme peut s'utiliser dans les 3 cas suivants:
  - Opérande de l'opérateur de cast
  - Opérande de l'opérateur sizeof
  - Déclaration d'un fonction

```
1  typedef int vect[3];  
2  typedef int *ptr_int;  
3  ....  
4  float fct(vect, ptr_int);  
5  sizeof(vect);  
6  (ptr_int) ad;
```

## C Nouveau type avec « typedef »

- Limitations de typedef:
  - Spécificateur de signe (**unsigned** / **signed**)
    - Utilisation à posteriori problématique
  - Qualificateurs (**const** / **volatile**)
    - Objet pointé vs objet pointant

## C Champ de bit

- **Utilité** : optimiser la mémoire, donner du sens à une structure de données ou décrire en termes de bits la structure d'une ressource matérielle de la machine

```

1 struct sr {
2     unsigned int r1 : 2;
3     unsigned int status : 2;
4     unsigned int : 1; /* inutilisé */
5     unsigned int mask : 3;
6     unsigned int : 3; /* inutilisé */
7     unsigned int crc : 4;
8 }
    
```

## C Champ de bit

- Représentation

```

1 struct sr {
2     unsigned int r1 : 2;
3     unsigned int status : 2;
4     unsigned int : 1; /* inutilisé */
5     unsigned int mask : 3;
6     unsigned int : 3; /* inutilisé */
7     unsigned int crc : 4;
8 }
    
```



## C Champ de bit

---

- Contraintes:
  1. Types acceptés: int, unsigned int, signed int.
  2. Dépend de l'endianness
  3. Un champ de bit déclaré comme étant de type int, peut en fait se comporter comme un signed int ou comme un unsigned int. Recommandé de déclarer les champs de bits comme étant de type unsigned int.
  4. Un champ de bits n'a pas d'adresse, on ne peut donc pas lui appliquer l'opérateur adresse de (&).

## C Enumération

---

- Pratique

```
#define noir      0
#define blanc    1
#define jaune    2
#define rouge    3   devient   enum couleur {noir, blanc, jaune, rouge, vert, bleu, violet};
#define vert     4
#define bleu     5
#define violet   6
```

## C Enumération

---

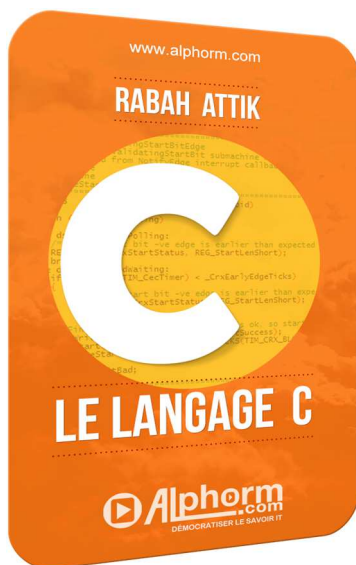
- Syntaxe
  - `enum [identificateur] { LISTE_d_enumerateurs }`
  - Les arguments sont des entiers,
  - Le premier énumérateur vaut 0
  - Les énumérateurs ne sont pas de l'value

## C Ce qu'on a couvert

---

- Nouveau type avec « typedef »
- champ de bit
- énumération





**Alphorm**.com

## Les structures

---

### Structures incomplètes C99 et anonymes en C11

Site : <http://formation.inge-tech.com>  
Blog : <http://www.alphorm.com/blog>  
Forum : <http://www.alphorm.com/forum>

**Rabah ATTIK**

Formateur et consultant indépendant  
Ingénierie système embarqué  
rabah.attik@inge-tech.com

**Formation Le langage C**

**alphorm.com™©**

## **C** Plan

---

- C90: Structure
- C99: Structure incomplète
- C11: Structure anonyme



**Formation Le langage C**

**alphorm.com™©**



## C C90: Structure

```
struct nom_du_modele
{
    type1 membre 1;
    type2 membre 2;
}
Puis
struct nom_du_modele nom_de_la_variable;
```

**OU**

```
struct nom_du_modele
{
    type1 membre 1;
    type2 membre 2;
} nom_de_la_variable;
```

```
struct
{
    int n;
    double d;
} s;
```

```
s.n = 1;
s.d = 1.432
```

- En C90 on peut omettre le nom du modèle lorsqu'il n'est pas utile

## C C99: Structure incomplète

- En C99 on peut déclarer une structure dont le dernier élément est un tableau donc on ne précise pas la dimension

```
struct stincomp
{
    int n;
    int ti[]; // dimension non précisée
};
```

**Exemple d'utilisation :**

```
int m = /* valeur définie */;
struct stincomp *p = malloc(sizeof (struct stincomp) + sizeof (int[m]));
```

L'objet pointé par p se comporte comme si p avait été déclaré de la manière suivante:

```
struct { int n; double d[m]; } *p;
```

Il y a des cas où l'offset de ti peut être corrompu

## C C11: Structure anonyme

- En C11 on peut omettre le nom du modèle et le nom de la variable

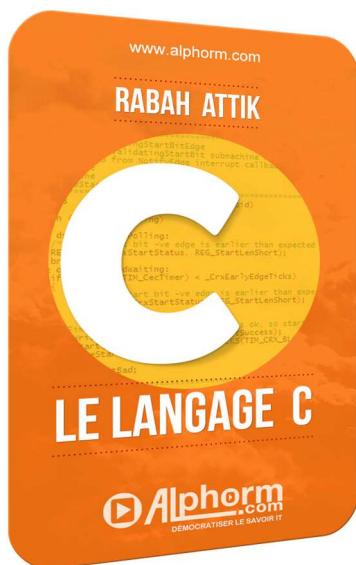
```
struct s1
{
    union // Union anonyme
    {
        struct s2 { int n; double d;} p1; // structure de nom p1
        struct s3 { long r; float x;}; // structure anonyme
    };
};

struct s1 s;
s.p1.n = 1; //OK
s.n = 1; // KO la structure incluant n'est pas anonyme
s.r = 1; // OK
```

## C Ce qu'on a couvert

- C90: Structure
- C99: Structure incomplète
- C11: Structure anonyme





**Alphorm**.com

## Les structures Listes chaînées

Site : <http://formation.inge-tech.com>  
Blog : <http://www.alphorm.com/blog>  
Forum : <http://www.alphorm.com/forum>

**Rabah ATTIK**

Formateur et consultant indépendant  
Ingénierie système embarqué  
[rabah.attik@inge-tech.com](mailto:rabah.attik@inge-tech.com)

Formation Le langage C

alphorm.com™©

### C Plan

- Pointeurs sur structures
- Liste chaînée
  - La liste chaînée: une structure de données
  - Liste chaînée simple
  - Liste chaînée double



Formation Le langage C

alphorm.com™©

## C Pointeurs sur structures

- structure

```
struct personne_structModel {  
    char* nom;  
    char* prenom;  
    int age;  
} sP, *p_sP;  
p_sP = &sP;  
//Ou  
p_sP = (struct personne_structModel*) malloc(sizeof(sP));
```

- p\_sP est un pointeur vers la structure s1 et contient l'adresse de la structure allouée sur le segment prévue pour l'allocation dynamique

## C Liste chaînée: une structure de données

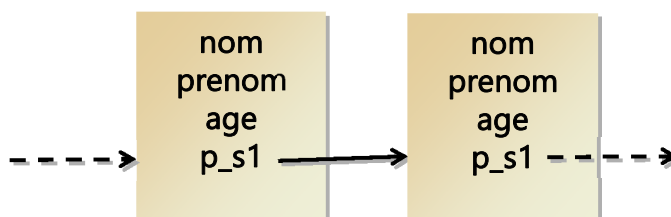
- Liste chaînée
  - Structure de données au même titre que les tableaux
  - Tableau / Liste / File / Pile / Arbre

## C Liste chaînée simple

- On constitue une liste simplement chaînée en déclarant un pointeur vers la même structure comme membre de celle-ci.

```
struct s1
{
    char * nom;
    char * prenom;
    int age;
    struct s1* p_s1;
} s;

s * p_s = NULL;
p_s = malloc(sizeof(s));
```

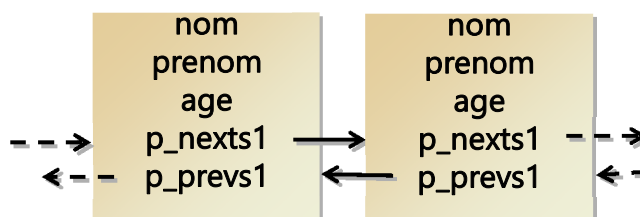


## C Liste chaînée double

- On constitue une liste doublement chaînée avec des structures possédant des pointeurs vers les successeurs et les prédécesseurs.

```
struct s1
{
    char * nom;
    char * prenom;
    int age;
    struct s1* p_nexts1;
    struct s1* p_prevs1;
} s;

s * p_s = NULL;
p_s = malloc(sizeof(s));
```

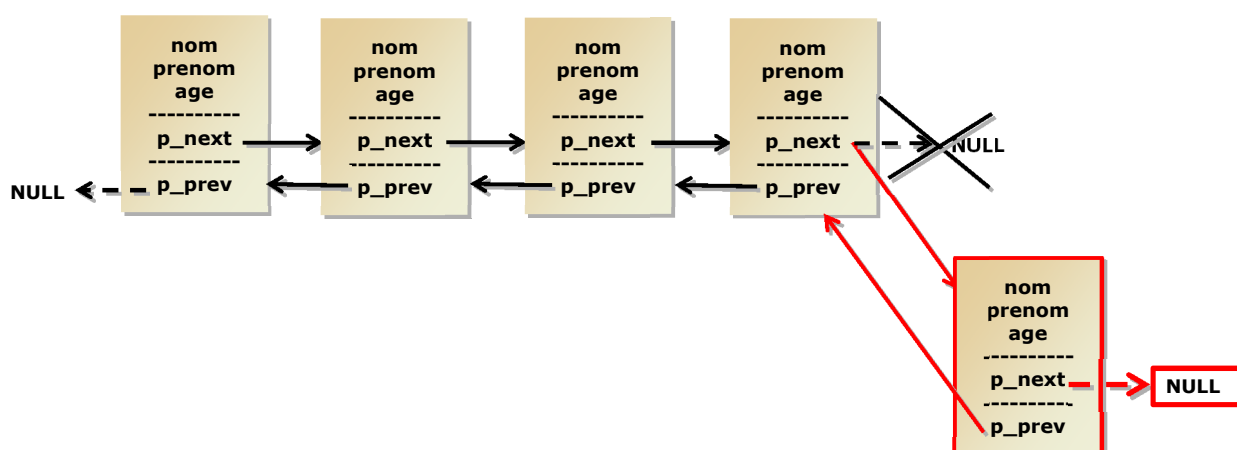


## C Primitives des listes chaînées

- Primitives des listes chaînées
  - Insertion
    - Au début / a la fin / dans la liste
  - Suppression
    - Selon position / valeur

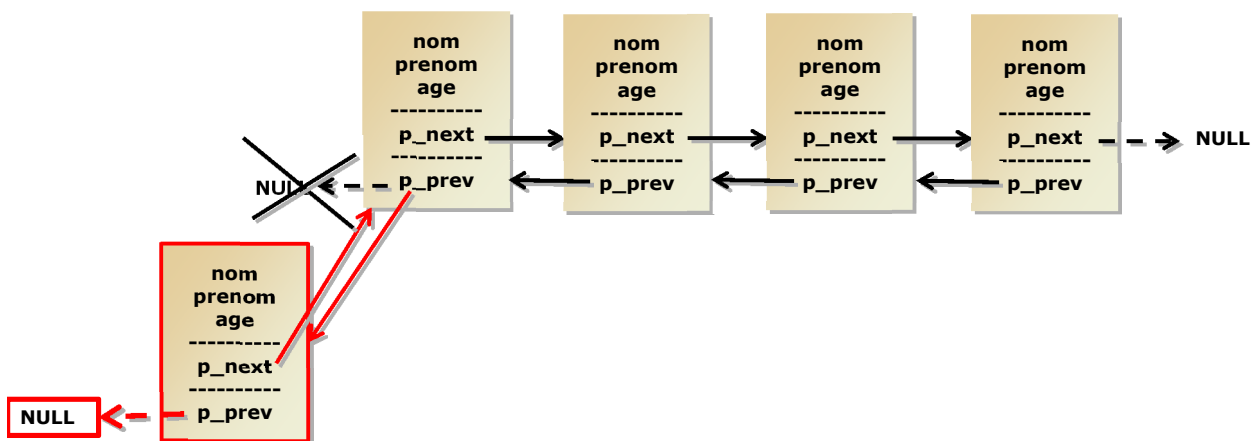
## C Primitives des listes chaînées

- Insertion: A la fin



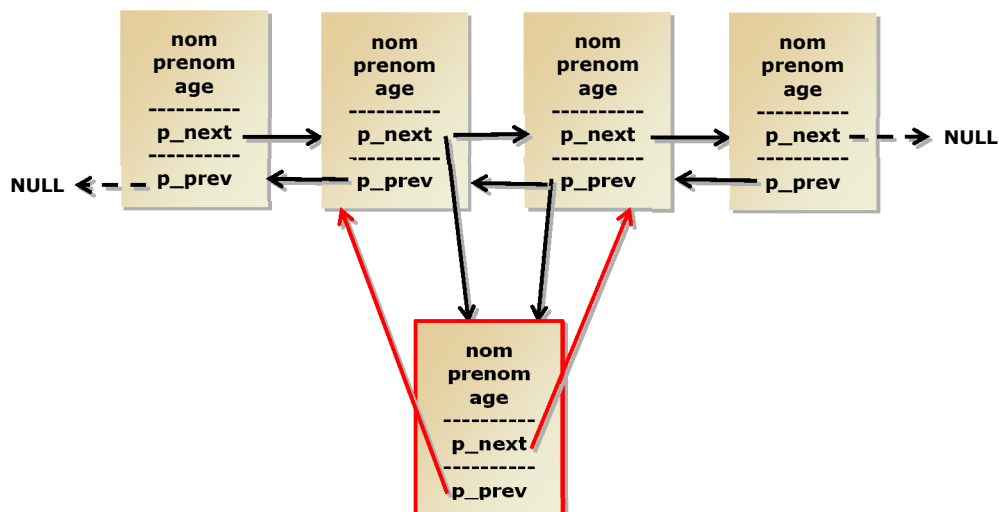
## C Primitives des listes chaînées

- Insertion: Au début



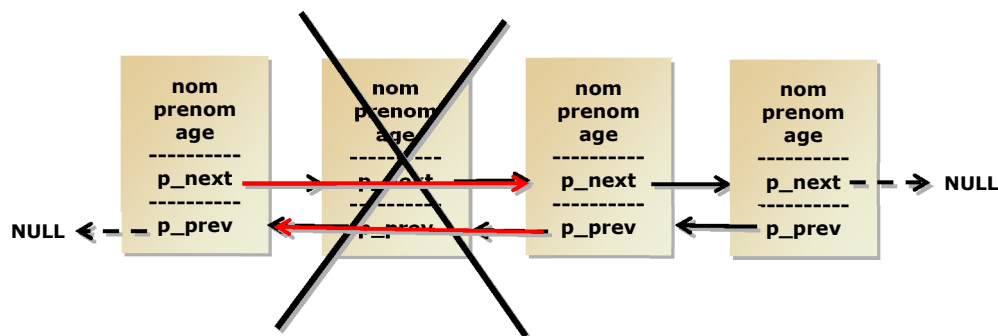
## C Primitives des listes chaînées

- Insertion: Dans la liste



## C Primitives des listes chaînées

- Suppression: Selon position / valeur

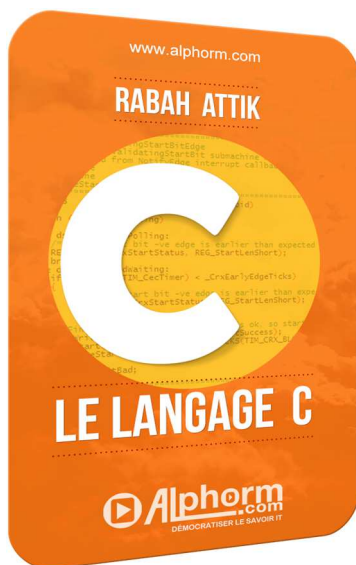


## C Ce qu'on a couvert

- Pointeur sur structures
- Listes chaînées
  - Alternatives aux tableaux
  - Données non concaténées, plus de souplesse
  - Plus gourmand en mémoire
  - Primitives des listes chaînées
  - Algorithmes seront abordés en « C avancé »







**Alphorm**.com

## Les fonctions

---

### Déclaration, définition et appel d'une fonction

Site : <http://formation.inge-tech.com>  
Blog : <http://www.alphorm.com/blog>  
Forum : <http://www.alphorm.com/forum>

**Rabah ATTIK**

Formateur et consultant indépendant  
Ingénierie système embarqué  
rabah.attik@inge-tech.com

**Formation Le langage C**

**alphorm.com™©**

## **C** Plan

---

- Les fonctions en C
- Déclaration d'une fonction
- Définition d'une fonction
- Appel d'une fonction



**Formation Le langage C**

**alphorm.com™©**

## C Les fonctions en C

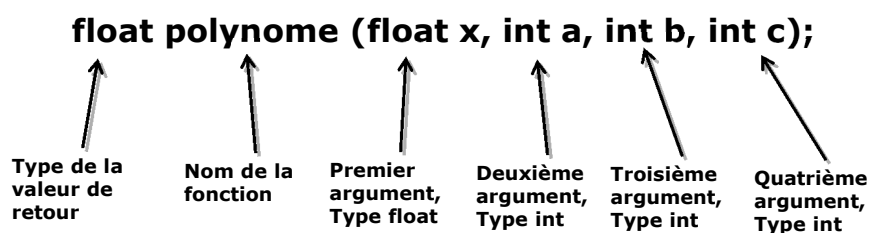
- Proche de la notion de module
- Mode de transmission : toujours par valeur
- Principe:



- Une fonction c'est
  - Déclarer
  - Définir
  - Appeler

## C Déclaration d'une fonction

- Déclaration d'une fonction ( le prototype de la fonction )



- Déclaration partielle

```
1 float polynome (float , int , int , int );
```

- Portée de déclaration de fonction

## C Définition d'une fonction

- Définition d'une fonction

```
1 float polynome (float x, int a, int b, int c)
2 {
3     /*On décrit ce que fait la fonction ici*/
4     float ret_val;
5     ret_val = a*x*x + b*x + c;
6     return ret_val;
7 }
```

En tête

Bloc d'instruction

Valeur de retour

- L'instruction «**return**»
  - Le type de la valeur de type quelconque sauf tableau ou fonction
  - Ne peut pas être qualifiée de const ou volatile, peut être un pointeur
- La définition tient lieu de déclaration dans la suite du même fichier source

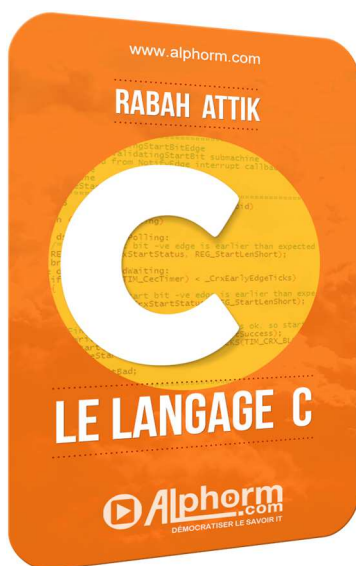
## C Appel d'une fonction

- Appel d'une fonction:
  - Possible en respectant les règles de portée
  - L'appel se fait sans le type retour en utilisant les arguments

## C Ce qu'on a couvert

---

- Les fonctions en C
- Déclaration d'une fonction
- Définition et appel d'une fonction



**Alphorm**.com

## Les fonctions Arguments et variables

---

Site : <http://formation.inge-tech.com>  
Blog : <http://www.alphorm.com/blog>  
Forum : <http://www.alphorm.com/forum>

**Rabah ATTIK**

Formateur et consultant indépendant  
Ingénierie système embarqué  
rabah.attik@inge-tech.com

## C Plan

- Rappel et suite sur les fonctions
- Mécanisme de transmission des arguments
- Cas des tableaux passés en argument
- Variables globales
- Variables locales



## C Rappel et suite sur les fonctions

- Un fonction c'est faire une déclaration, une définition puis appel
  - Suit les règles de portée de déclaration liées aux variables
  - Argument muet et argument effectif

```
int f(int i);  
int f(int i)  
{  
    ....  
}  
int main()  
{  
    int i;  
    f(i);  
}
```

- « const » pour les arguments
- Non concordance entre arguments
  - Conversion implicite

- La notion de classe d'allocation devient classe de mémorisation
  - static
- Les qualifieurs pour la valeur de retour n'ont plus de sens
  - Cas des pointeurs: Différencier valeur de retour et valeur pointée

## C Mécanisme de transmission des arguments

- Arguments : toujours transmis par valeur
  - La fonction reçoit une copie de la valeur de l'argument effectif

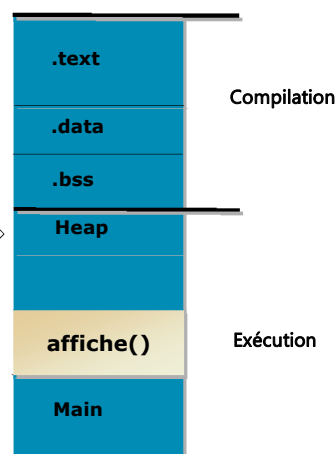
```

1  #include <stdio.h>
2  void affiche(int p);
3
4  int main()
5  {int n =5;
6   printf(« n avant: %d\n »,n);
7   affiche(n);
8   printf(« n après: %d\n »,n);
9  }
10
11 void affiche(int p)
12 {printf(« p avant: %d\n »,p);
13  p *=3;
14  printf(« p après: %d\n »,p);
15 }
    
```

*n avant: 5*  
*p avant: 5*  
*p après: 15*  
*n après: 5*

Allocation dynamique  
sur le « tas » par  
malloc

Segment de pile  
Pile d'exécution



## C Mécanisme de transmission des arguments

- Tableau non transmis par valeur
- Structure transmise par valeur
  - Inconvénients en terme de temps et de mémoire
  - On peut préférer manipuler l'adresse de la structure

## C Mécanisme de transmission des arguments

- Limite du passage par valeur

```
#include <stdio.h>
void echange(int a, int b);

int main()
{
    int n = 10, p = 20;
    printf(« Avant: %d %d\n », n, p);
    echange(n, p);
    printf(« Après: %d %d\n », n, p);
}

void echange( int a, int b)
{
    int c;
    printf(« Echange avant: %d %d\n », n, p);
    c = a;
    a = b;
    b = c;
    printf(« Echange après: %d %d\n », n, p);
}
```

Avant: 10 20  
Echange avant: 10 20  
Echange après: 20 10  
Après: 10 20

- **Solution** : pointeur / return / variable globale

## C Mécanisme de transmission des arguments

- Passage par adresse

```
#include <stdio.h>
void echange(int * a, int * b);

int main()
{
    int n = 10, p = 20;
    printf(« Avant: %d %d\n », n, p);
    echange(&n, &p);
    printf(« Après: %d %d\n », n, p);
}

void echange (int * a, int * b)
{
    int c;
    printf(« Echange avant: %d %d\n », n, p);
    c = *a;
    *a = *b;
    *b = c;
    printf(« Echange après: %d %d\n », n, p);
}
```

Avant: 10 20  
Echange avant: 10 20  
Echange après: 20 10  
Après: 20 10

## C Cas des tableaux passés en argument

- Cas particulier en C à cause de la corrélation pointeur / tableau
- `f(t1)` équivaut à `f(&t1[0])`
- Différence entre tableau en argument muet et en argument effectif
  - Lvalue
  - Effectif: `f(t1)` ou `f(&t[0])` ou `f(&t[i])` (pour traiter un sous tableau)
  - Muet: `int f1(int t[])` OU `int f1(int t[5])` OU `int f1(int *t)`

## C Variables globales

- Une variable globale (VG) est en théorie accessible, à toutes les fonctions définies ou non dans le même fichier source.
- Cas représentatifs de l'utilisation des VG
  - Accès à une VG déclarée dans le même fichier source
  - Accès à une VG déclarée dans un autre fichier source
  - Restriction d'une VG déclarée dans un même fichier source



## C Variables globales

Déclaration	Définition Redéclaration
Portée	Limitée au fichier source
Lien	Externe par défaut
Classe d'allocation	Statique
Initialisation	Par défaut à une valeur nulle Explicitement avec des expressions constantes
Inconvénients	Risques => réutilisation

=> **Démo**

## C Variables locales

- Variable locale à un bloc ou à une fonction
- Exception : variable globale définie par « extern » ailleurs

Portée	Le bloc
Classe d'allocation	<i>auto</i> par défaut Peut être <i>static</i> ou <i>register</i>
Initialisation	Pas d'initialisation par défaut Effectuée à chaque entrée dans la fonction sauf pour la classe <i>static</i>

## C Variables locales

---

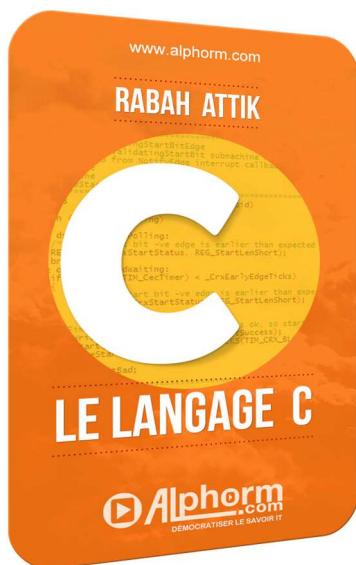
- Les trois classes d'allocation
  - *auto*
  - *static*
  - *register*

## C Ce qu'on a couvert

---

- Rappel fonction, déclarations et valeur de retour
- Mécanisme de transmission d'argument
  - Par valeur
  - Par adresse
- Cas des tableaux en argument
- Variables globales et locales





**Alphorm**.com

## Les fonctions

---

# Pointeurs sur fonctions

Site : <http://formation.inge-tech.com>  
Blog : <http://www.alphorm.com/blog>  
Forum: <http://www.alphorm.com/forum>

**Rabah ATTIK**

Formateur et consultant indépendant  
Ingénierie système embarqué  
rabah.attik@inge-tech.com

**Formation Le langage C**

**alphorm.com™©**

## **C** Plan

---

- Pointeur sur fonction
  - Utilité
  - Déclaration
  - Affectation
- Utilisation
  - Paramétrage d'appel de fonction
  - Transmission de fonction en argument



**Formation Le langage C**

**alphorm.com™©**

## C Pointeur sur fonction

---

- Utilité :
  - Paramétrer l'appel d'une fonction
  - Transmettre en argument une fonction à une autre fonction

## C Pointeur sur fonction

---

- Déclaration: **int (\* ptr\_fc)(double, int);**
  - ptr\_func pointe sur une fonction a deux arguments de type double et int et renvoyant un int

**(\* ptr\_fc)(double, int) est un int**

**(\* ptr\_fc) est une fonction recevant en argument un double et un int et renvoyant un int**

**\*ptr\_fc est une fonction recevant en argument un double et un int et renvoyant un int**

**ptr\_fc est un pointeur sur une fonction recevant en argument un double et un int et renvoyant un int**

- Parenthèses indispensables **int \* ptr\_fc(double, int);**

## C Pointeur sur fonction

- Affectation:

```
int f1(double, int);
int f2(float);

int (*ptrf1) (double, int);
int (*ptrf2) (double, int);

ptrf1 = f1; // correct équivaut à ptrf1 = &f1
ptrf2 = ptrf1; // correct

ptrf1 = f2; // incorrect
```

## C Utilisation

- Appel d'un fonction par le biais d'un pointeur

```
int i;
void func1(void);
void func2(void);
void (*fct[]) (void) = {func1, func2};
int sequence[] = {1,2,1,1,2,2};
for(i = 0 ; i < sizeof(sequence)/sizeof(sequence[1]) ; i++)
{
    fct[sequence[i]-1] (); // fct[sequence[i]-1] ne fait rien
}

void func1(void)
{
    printf(«Action 1\n»);
}

void func2(void)
{
    printf(«Action 2\n»);
}
```

## C Utilisation

- Transmission de fonction en argument

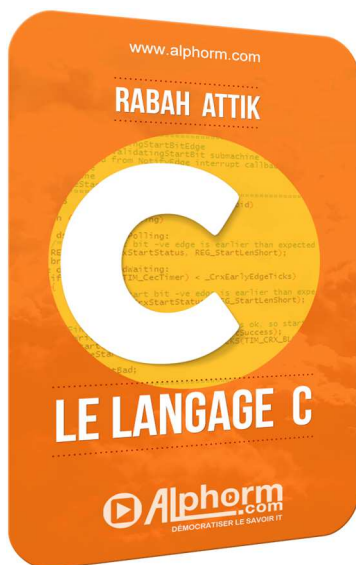
```
float integ (float (*f) (float), int min, int max) void func2(void);  
float funct1(float);  
float funct2(float);  
...  
res1 = integ (funct1, ...);  
res2 = integ (funct2, ...);
```

- Demo

## C Ce qu'on a couvert

- Pointeur sur fonction
  - Déclaration, affectation
- Utilisation:
  - Appel / argument





**Alphorm**.com

## Compilation séparée Chaîne de production

Site : <http://www.alphorm.com>  
Blog : <http://blog.alphorm.com>  
Forum : <http://forum.alphorm.com>

**Rabah ATTIK**  
Formateur et Consultant indépendant  
Ingénierie système embarqué

Formation Le langage C

alphorm.com™©

### C Plan

- Rappel et présentation de la compilation séparée
- Présentation de la chaîne de production: cas simple
  - Pré-compilation & compilation
  - Edition de liens
  - Chargement en RAM
- Cas de plusieurs fichiers



Formation Le langage C

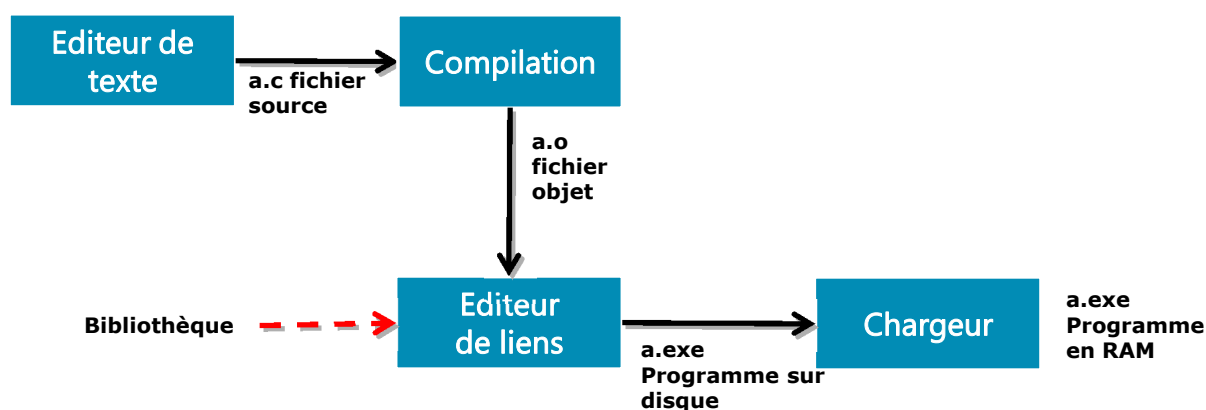
alphorm.com™©

## C Présentation de la chaîne de production

- Précompilation & compilation
  - Analyse des macros et transformation du fichier source en fichier objet
- Edition de liens
  - Génération du fichier binaire
- Chargement
  - Gérer par l'OS, chargement en RAM

## C Présentation de la chaîne de production: cas simple

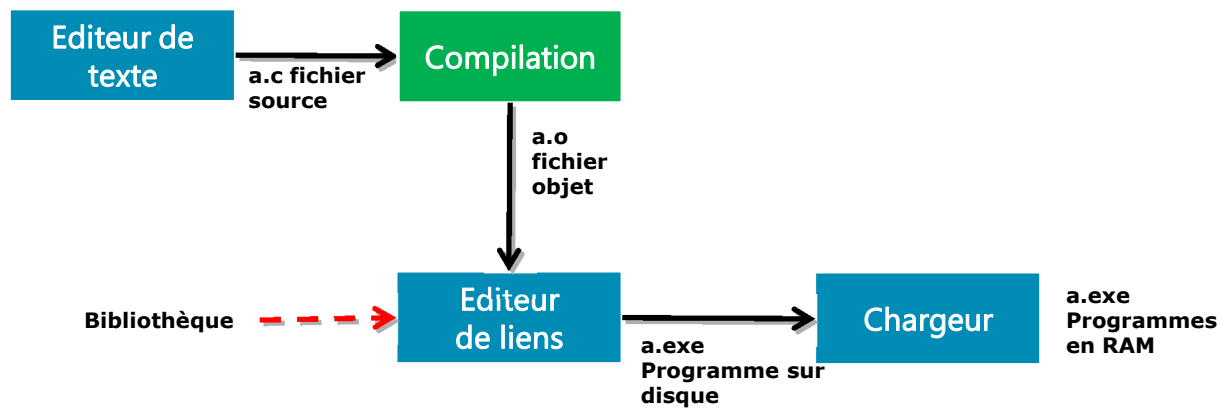
- Chaîne de production





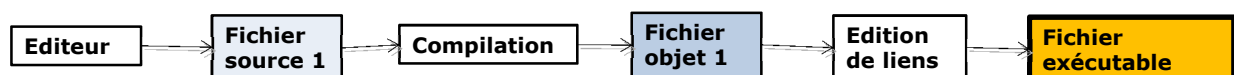
## C Compilation

- Compilation



## C Compilation

- Chaîne élémentaire de compilation

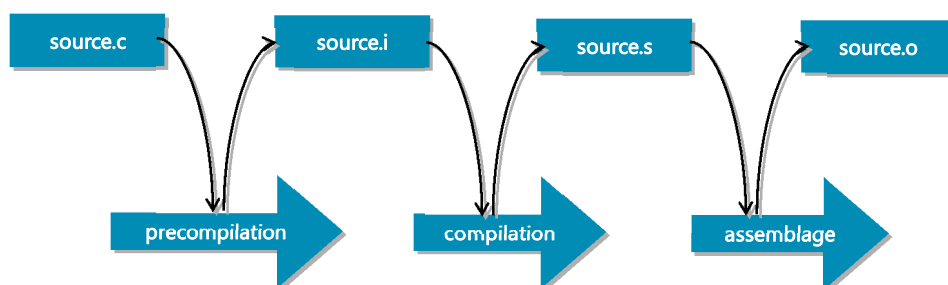


## C Compilation

- Précompilation & compilation
  - Analyse des macros et transformation du fichier source en fichier objet
  - Les calculs d'adresses ne sont pas résolus
  - Plus en détails, le compilateur :
    - analyse le programme au niveau syntaxique et sémantique,
    - traduit le langage de haut niveau en langage machine,
    - affecte des adresses aux variables

## C Compilation

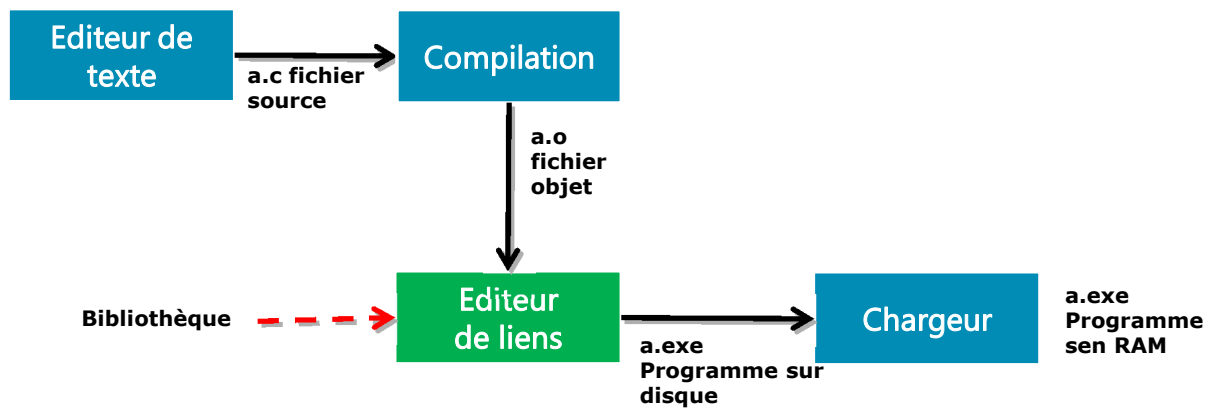
- Mais encore ?



## Démo

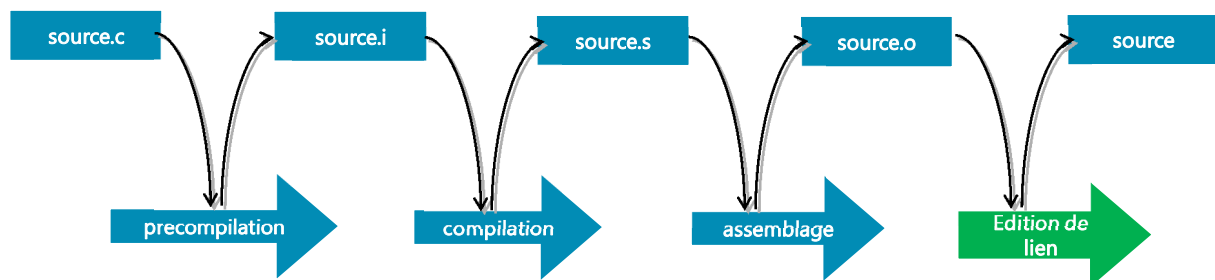
## C Edition de liens

- Edition de liens



## C Edition de liens

- Mais encore ?

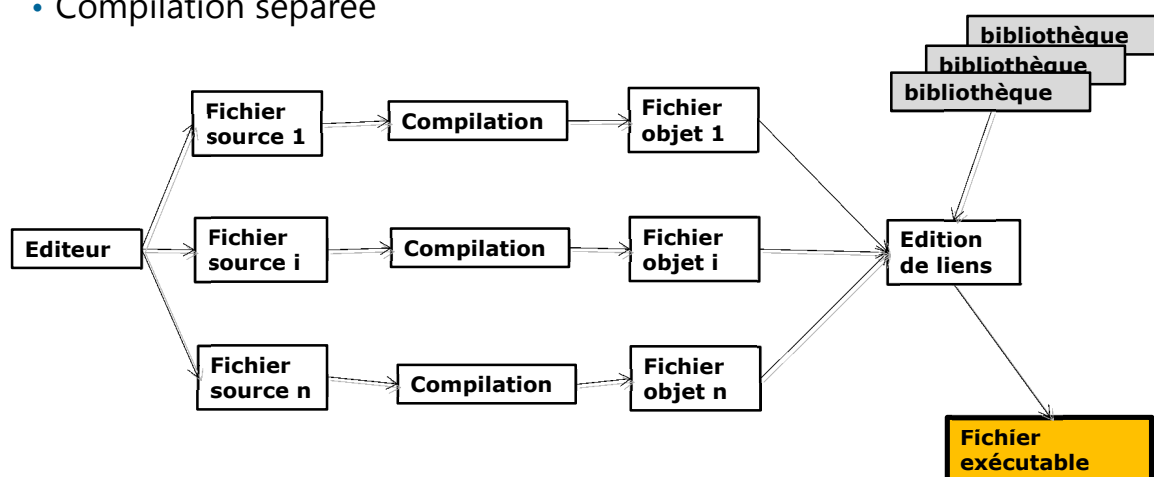


## C Edition de liens

- Edition de liens (linker)
  - **Rôle** : Produire un module chargeable. (Générer du fichier binaire)
  - Pour cela, le linker :
    - Construit une table qui indique le nom, la taille et la longueur de tous les modules objet,
    - Affecte une adresse de chargement à chaque module objet,
    - Effectue la translation en modifiant les instructions qui contiennent une référence mémoire,
    - Résout les références externes en insérant l'adresse des procédures à l'endroit où elles sont appelées

## C Cas de plusieurs fichiers

- Compilation séparée



## C Cas de plusieurs fichiers

---

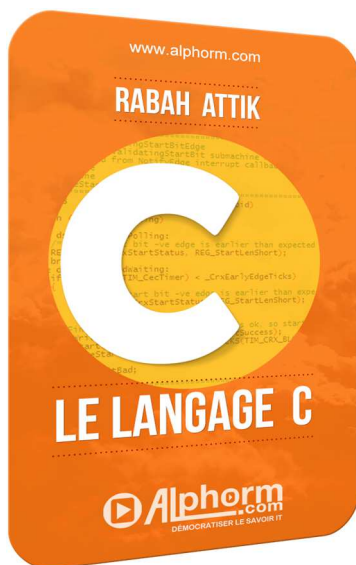
- La compilation séparée des fichiers fait l'objet d'une analyse approfondie du compilateur
  - On parle de classe d'allocation pour les variables

## C Ce qu'on a couvert

---

- La chaîne de production
- La compilation
- L'édition de lien





**Alphorm**.com

## Compilation séparée L'outil Makefile

Site : <http://www.alphorm.com>  
Blog : <http://blog.alphorm.com>  
Forum : <http://forum.alphorm.com>

**Rabah ATTIK**  
Formateur et Consultant indépendant  
Ingénierie système embarqué

Formation Le langage C

alphorm.com™©

### C Plan

- La compilation séparée sans Makefile
- Makefile : Principe
- Syntaxe
  - Cibles, dépendances et commandes
  - Variables
    - Personnalisées
    - Internes
- Règles d'inférence
- La cible PHONY
- Génération des fichiers objets et sources
- Gestion des conditions



Formation Le langage C

alphorm.com™©

## C Principe

- Fichier utilisé par le programme **make** pour réaliser un ensemble d'actions
- Multitude d'utilitaires (*gmake, nmake, tmake, ...*)
- Non normalisé
  - Existe des problèmes d'incomptabilité
- GNU make

## C Cible, dépendances, commandes

- cible: dependance  
commandes  
  
prog: main.o affichage.o  
    gcc -o prog main.o affichage.o  
  
main.o: main.c  
    gcc -o main.o -c main.c -Wall -ansi -pedantic  
  
affichage.o: affichage.c  
    gcc -o affichage.o -c affichage.c -Wall -ansi -pedantic

## C Variable personnalisée

- NOM=VALEUR => Utilisation: \$(NOM)

CC=gcc

CFLAGS=-Wall -ansi -pedantic

prog: main.o affichage.o

\$(CC) -o prog main.o affichage.o

main.o: main.c

\$(CC) -o main.o -c main.c \$(CFLAGS)

affichage.o: affichage.c

\$(CC) -o affichage.o -c affichage.c \$(CFLAGS)

clean:

rm -rf \*.o

mrproper: clean

rm -rf prog

## C Variable interne

- Il existe plusieurs variables internes au **Makefile**:

\$@	Le nom de la cible
-----	--------------------

\$<	Le nom de la première dépendance
-----	----------------------------------

\$^	La liste des dépendances
-----	--------------------------

\$?	La liste des dépendances plus récentes que la cible
-----	---

\$*	Le nom du fichier sans suffixe
-----	--------------------------------



## C Variable interne

- Ajout des variables internes :

CC=gcc

CFLAGS=-Wall -ansi -pedantic

LDFLAGS=

prog: main.o affichage.o

\$(CC) -o \$@ \$^ \$(LDFLAGS)

main.o: main.c

\$(CC) -o main.o -c main.c \$(CFLAGS)

affichage.o: affichage.c

\$(CC) -o affichage.o -c affichage.c \$(CFLAGS)

clean:

rm -rf \*.o

mrproper: clean

rm -rf prog

## C Règle d'inférence

- Règles génériques : Ex: %.o: %.c

CC=gcc

CFLAGS=-Wall -ansi -pedantic

LDFLAGS=

prog: main.o affichage.o

\$(CC) -o \$@ \$^ \$(LDFLAGS)

main.o: main.c

\$(CC) -o main.o -c main.c \$(CFLAGS)

devient:

%.o: %.c

\$(CC) -o \$@ -c \$< \$(CFLAGS)

...

## C La cible .PHONY

- Reconstruire les dépendances de la règle PHONY

CC=gcc

CFLAGS=-Wall -ansi -pedantic

LDFLAGS=

prog: main.o affichage.o

\$(CC) -o \$@ \$^ \$(LDFLAGS)

main.o: main.c

\$(CC) -o main.o -c main.c \$(CFLAGS)

main.o: affichage.h

%.o: %.c

\$(CC) -o \$@ -c \$< \$(CFLAGS)

**.PHONY: clean mrproper**

clean:

rm -rf \*.o

mrproper: clean

rm -rf prog

## C Génération de fichier objets et sources

OBJ= \$(SRC:.c=.o)

CC=gcc

CFLAGS=-Wall -ansi -pedantic

LDFLAGS=

**SRC= hello.c main.c**

**OBJ= \$(SRC:.c=.o)**

prog: main.o affichage.o

\$(CC) -o \$@ \$^ \$(LDFLAGS)

main.o: main.c

\$(CC) -o main.o -c main.c \$(CFLAGS)

affichage.o: affichage.c

\$(CC) -o affichage.o -c affichage.c \$(CFLAGS)

...

## C Génération de fichier objets et sources

```
OBJ= $(SRC:.c=.o)
CC=gcc
CFLAGS=-Wall -ansi -pedantic
LDFLAGS=
SRC= hello.c main.c => ($wildcard *.c)
OBJ= $(SRC:.c=.o)

prog: main.o affichage.o
    $(CC) -o $@ $^ $(LDFLAGS)
main.o: main.c
    $(CC) -o main.o -c main.c $(CFLAGS)

affichage.o: affichage.c
    $(CC) -o affichage.o -c affichage.c $(CFLAGS)
...
```

## C Commande silencieuse

```
Utilisation de @

CC=gcc
CFLAGS=-Wall -ansi -pedantic
LDFLAGS=

prog: main.o affichage.o
    @$(CC) -o $@ $^ $(LDFLAGS)
main.o: main.c
    @$(CC) -o main.o -c main.c $(CFLAGS)

affichage.o: affichage.c
    @$(CC) -o affichage.o -c affichage.c $(CFLAGS)
clean:
    rm -rf *.o
mrproper: clean
    rm -rf hello
```

## C Gestion des conditions

Utilisation de @

```
DEBUG=yes
CC=gcc
ifeq ($(DEBUG),yes)
    CFLAGS=-W -Wall -ansi -pedantic -g
Else
    CFLAGS=-W -Wall -ansi -pedantic
endif
LDFLAGS=

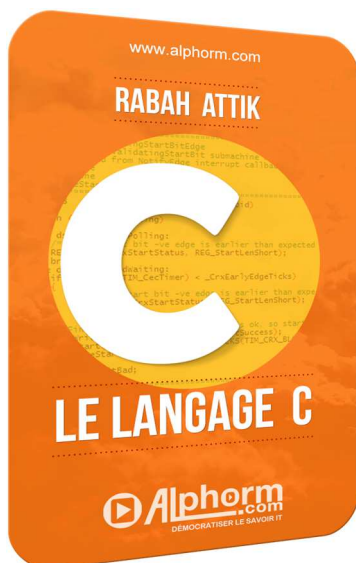
prog: main.o affichage.o
    @$(CC) -o $@ $^ $(LDFLAGS)
main.o: main.c
    @$(CC) -o main.o -c main.c $(CFLAGS)
```

...

## C Ce qu'on a couvert

- Problématique de la gestion en ligne de commande
- Makefile





**Alphorm**.com

## Compilation séparée Configuration mémoire d'un programme

Site : <http://www.alphorm.com>  
Blog : <http://blog.alphorm.com>  
Forum : <http://forum.alphorm.com>

Rabah ATTIK  
Formateur et Consultant indépendant  
Ingénierie système embarqué

Formation Le langage C

alphorm.com™©

### C Plan

- Les différents segments d'un programme
  - .text
  - .bss
  - .data
  - heap
  - stack



Formation Le langage C

alphorm.com™©

## C Segment de texte

---

- Contient les instructions à exécuter
- Lecture seule

## C Segment donnée

---

- Partie du segment de données
- Contient les variables globales ou statiques initialisées
- Accès en écriture

## C Segment BSS

---

- Partie du segment de données
- Contient les variables globales ou statiques non initialisées
- Accès en écriture

## C Pile d'exécution

---

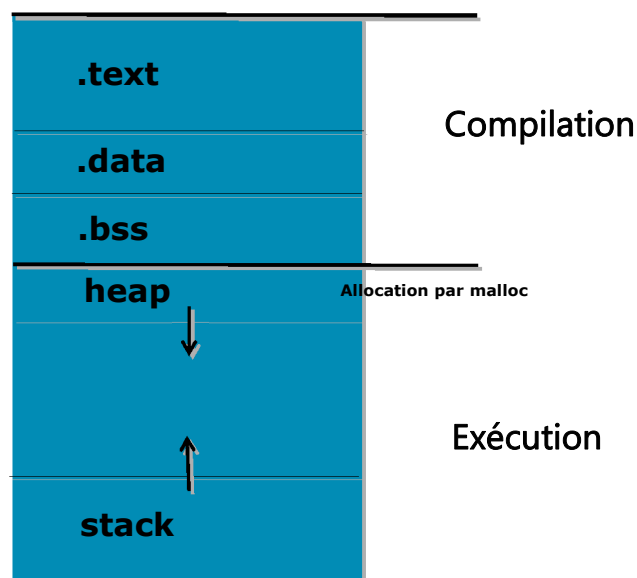
- La pile est une structure LIFO.
- Généralement située dans la plus haute zone de la mémoire
- **Compteur ordinal** : Registre de pointeur de pile qui pointe sur le haut de la pile
  - Modifié à chaque fois qu'une valeur est empilée ou dépilée
- **Trame de pile** : L'ensemble des valeurs empilées par un appel à une fonction.

## C Segment d'allocation dynamique

- Allocation dynamique par les fonctions de type m/c/re/alloc
- Doit être libérée
- Objet de fuite de mémoire
- Partagé par toutes les bibliothèques et les modules chargés dynamiquement par un processus

## C Illustration

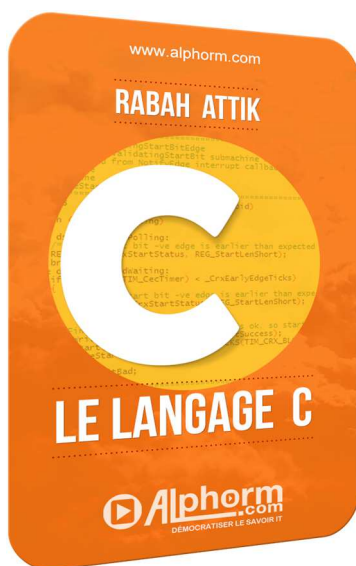
- Illustration





## C Ce qu'on a couvert

- Les différents segments d'un programme
  - Segment de texte
  - Segment bss
  - Segment de données
  - La pile d'exécution
  - Le segment d'allocation dynamique



**Alphorm**.com

## Compilation séparée Classe d'allocation des variables

Site : <http://formation.inge-tech.com>  
Blog : <http://www.alphorm.com/blog>  
Forum : <http://www.alphorm.com/forum>

**Rabah ATTIK**

Formateur et consultant indépendant  
Ingénierie système embarqué  
rabah.attik@inge-tech.com

## C Plan

---

- Les classes d'allocation (ou classe de mémorisation) :
  - Automatique -> mot clé auto
  - Statique -> mot clé static
  - Registre -> mot clé register
  - Externe -> mot clé extern



## C Classe d'allocation automatique

---

- Portée à l'intérieur d'un bloc de code ou d'un bloc de fonction
- Allouée sur la pile
- Les arguments explicites sont copiés sur la pile et sont en classe d'allocation auto

## C Classe d'allocation statique

---

- 3 cas :
  1. Variable globale de classe statique
  2. Fonction de classe statique
  3. Variable locale de classe statique

Dans le cas 1. et 2. => Portée sur un module objet

Dans le cas 3. La variable locale garde sa valeur d'un appel à l'autre car la variable n'est pas allouée sur la pile mais sur le segment **.DATA** ou **.BSS** du processus

## C Classe d'allocation registre

---

- Cas d'optimisation :
  - On indique au système d'allouer la variable sur un registre du cache **SI C'EST POSSIBLE**

## C Classe d'allocation externe

---

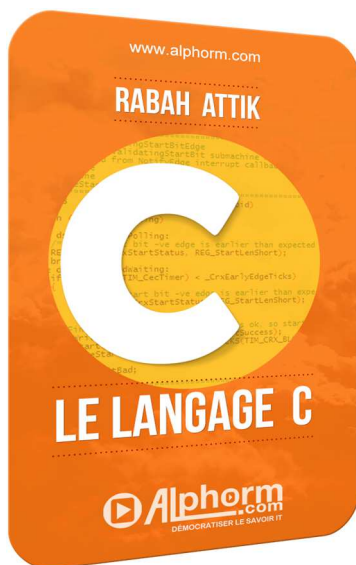
- Variable globale redéfinie dans un module objet :
  - Problématique multifichiers

## C Ce qu'on a couvert

---

- Les classes d'allocation:
  - Automatique
  - Statique
  - Registre
  - Externe





**Alphorm**.com

## Le préprocesseur

---

### Préprocesseur et directives

Site : <http://formation.inge-tech.com>  
Blog : <http://www.alphorm.com/blog>  
Forum : <http://www.alphorm.com/forum>

**Rabah ATTIK**

Formateur et consultant indépendant  
Ingénierie système embarqué  
rabah.attik@inge-tech.com

Formation Le langage C

alphorm.com™©

## C Plan

---

- Le préprocesseur
- Les directives
- Le fonctionnement du préprocesseur
- #define et #undef
  - Symbole
  - Macros
- #include
- Macros prédéfinies



Formation Le langage C

alphorm.com™©

## C Le préprocesseur

---

- Traduction du fichier source en langage machine = 2 parties
  1. Préprocesseur
  2. Compilation
- Préprocesseur = traitement du texte du fichier source, interprétation d'instructions particulières appelées des directives
  - gcc -E -o source.i source.c

## C Les directives

---

- Une directive :
  - Commence par '#'
  - Tiennent compte des lignes (« \ » pour passer à une autre ligne)

## C Le fonctionnement du préprocesseur

- L'expansion :
  - Identification du symbole ou des paramètres de macros
  - Expansion éventuelle de paramètres de macros
  - Substitution d'un symbole ou des paramètres d'une macro
  - Répétition du processus d'expansion

## C Les directives

- Les directives

Catégorie	Directives	Rôle
Définition de symbole ou de macros	#define #undef	Définit un symbole ou une macro Annule une définition de symbole ou de macro
Compilation conditionnelle	#if #else #endif #elif #ifdef #ifndef	Teste la valeur d'une expression Condition fausse Termine les instruction #if, #ifdef, #ifndef Permet de clôturer une suite #else #if Teste l'existence d'un symbole Teste l'inexistence d'un symbole
Inclusion de fichier source	#include	
Divers	#line #error #pragma #	Impose un numéro de ligne dans le fichier source Affiche un message d'erreur Spécifique à une implémentation Directive vide

## C La directives #define

- Permet de définir des symboles et des macros
  - Symboles

```
1 #define NMAX 5
2 #define TAILLE NMAX+1
3
4 #define PI 3.14.159265
5 #define TRUE 1
6 #define FALSE 0
7 #define MES "Hello"
8 #define FL '\n'
```

- Sera remplacée par le texte par le préprocesseur, respecte les contraintes de syntaxe des identificateurs

## C #define

- Symboles

```
1 #define hello printf('Hello')
2
3 hello; /*remplacé par print('hello');*/
4
```

- Macro = symbole plus élaboré



## C #define

- Possibilité d'utiliser un/des paramètre(s)

```
1 #define carre(a) a*a
2 #define diff(a,b) a-b
```

- Effet de bord

```
1 carre(a+b); /* donne a+b*a+b */
```

- Etre rigoureux

```
1 #define carre(a) (a)*(a)
```

## C #undef

- Utilité :
  - Annule une définition de symbole ou de macros

```
1 #define NMAX 1000
2 ...
3 #undef NMAX
4 ...
5 #define NMAX 2000
6
```

## C #include

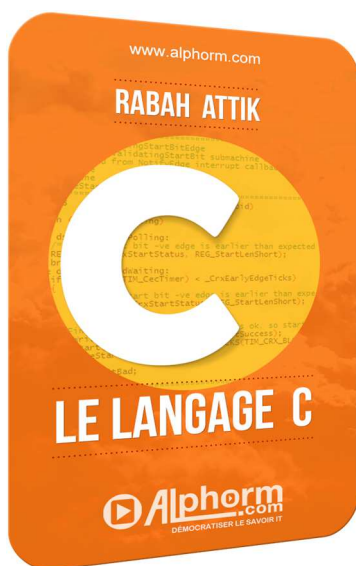
- Utilité :
  - Incorporé dans le fichier source un texte provenant d'un autre fichier
  - Pas de restrictions par rapport aux macros
  - Utilisé pour les fichiers d'entête
  - Attention aux inclusions multiples
    - #ifndef

## C Macros prédéfinies

Symbole	Signification	Remarques
__LINE__	Numéro de ligne à l'intérieur du fichier source: constante décimale	Peut être modifié par la directive #line
__FILE__	Nom de fichier source: chaîne de caractères constante	La norme ne précise pas si le chemin fait partie de ce nom
__DATE__	Date à laquelle a lieu la compilation: chaîne de la forme « mmm jj aaa »	La valeur reste la même pour toute la durée de la compilation
__TIME__	Heure à laquelle a lieu la compilation: chaîne de la forme hh:mm:ss	La valeur reste la même pour toute la durée de la compilation
__STDC__	Constante 1 pour indiquer que l'implémentation est conforme à la norme ANSI	

## C Ce qu'on a couvert

- Le préprocesseur
- Les directives
- #define et #undef
  - Symbole
  - Macros
- #include
- Macros prédéfinies



**Alphorm**.com

## Le préprocesseur

### Les marqueurs # et ##

Site : <http://formation.inge-tech.com>  
Blog : <http://www.alphorm.com/blog>  
Forum : <http://www.alphorm.com/forum>

**Rabah ATTIK**

Formateur et consultant indépendant  
Ingénierie système embarqué  
rabah.attik@inge-tech.com

## C Plan

---

- Opérateur de conversion en chaîne : #
- Notion de token
- Opérateur de concaténation de tokens : ##
- Conséquences d'utilisation



## C Opérateur de conversion en chaîne: #

---

- **Problématique**
  - La substitution de paramètres ou de symbole ne se fait pas à l'intérieur des chaînes de caractères

```
1 #define affiche(a) printf(''valeur de a : %d\n'', a)
```

- Opérateur #

```
1 #define affiche(a) printf(''valeur de'' #a '' : %d\n'', a)
```

## C Notion de token

- Token => Un programme peut se décomposer en une succession d'éléments appelés token
- Notion indispensable pour le compilateur mais pas pour le programmeur
- Token reconnu par le préprocesseur et par le compilateur
- Décomposition en token, exemple:
  - `x1 + 2` conduit aux tokens `x1` `+` `et` `é`
  - `x+++3` conduit aux tokens `x++` `+` `3`
  - => le préprocesseur et le compilateur recherche la longue séquence de caractère possible qui soit un token

## C Opérateur de concaténation de tokens: ##

- **Problématique**
  - `#define machin truc` => `machin1` ne produira pas `truc1` mais sera un identificateur à part entière
- Opérateur ##
  - `#define concatenate(a,b) a ## b`
  - `concatenate(machin,1)` produira `machin1`
  - Précédé et suivi d'un token

## C Conséquences

- # et ## bloquent l'expansion des paramètres effectifs

#

```
#define NMAX 50  
#define chaine(x) #x
```

chaine(NMAX) fournit « NMAX »  
et non « 50 »

##

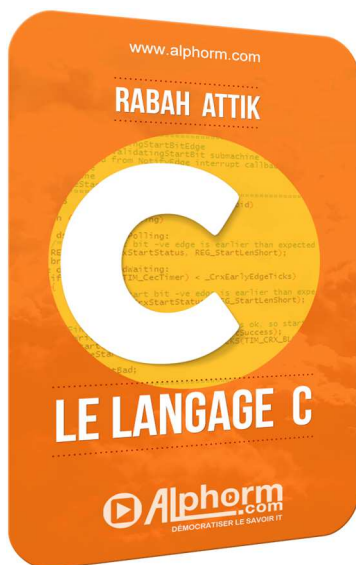
```
#define VERSION 3  
#define nomfich(x) fichier ## x
```

nomfich(VERSION) fournit  
« fichierVERSION » et non « fichier3 »

## C Ce qu'on a couvert

- Opérateur de conversion en chaîne: #
- Notion de token
- Opérateur de concaténation de tokens: ##
- Conséquence d'utilisation





**Alphorm**.com

## Le préprocesseur

---

## La compilation conditionnelle

Site : <http://formation.inge-tech.com>  
Blog : <http://www.alphorm.com/blog>  
Forum : <http://www.alphorm.com/forum>

**Rabah ATTIK**

Formateur et consultant indépendant  
Ingénierie système embarqué  
rabah.attik@inge-tech.com

Formation Le langage C

alphorm.com™©

### **C** Plan

---

- Compilation conditionnelle par symboles
- Compilation conditionnelle par expressions
- Imbrications de directives de compilation conditionnelle
- Cas d'utilisation



Formation Le langage C

alphorm.com™©

## C Compilation conditionnelle par symboles

- Exemple

```
#define DEBUG
...
#ifdef DEBUG
    instruc_1; #else
    instruc_2;
#endif
...
#ifdef DEBUG
    instruc_3; #endif
```

Ou

```
#define DEBUG
...
#ifndef DEBUG
    instruc_2;
#else
    instruc_1;
#endif
...
#ifndef DEBUG
    instruc_3;
#endif
```

## C Compilation conditionnelle par expressions

- Exemple

```
#define COND 1
...
if COND == 1
    ...
endif
if COND == 2
    ...
endif
```

Ou

```
#define COND 1
...
if COND == 1
    ...
elif COND == 2
    ...
endif
```



## C Compilation conditionnelle par expressions

- Exemple

```
#if defined(COND)
    instruc_1;
#else
    instruc_2;
#endif
```

## C Imbrications de directives

- Exemple

```
#if COND_1
    instruc_1;
#else
    instruc_2;
#endif
#if COND_2
    instruc_3;
#else
    instruc_4;
#endif
#endif
```

## C Cas d'utilisation

---

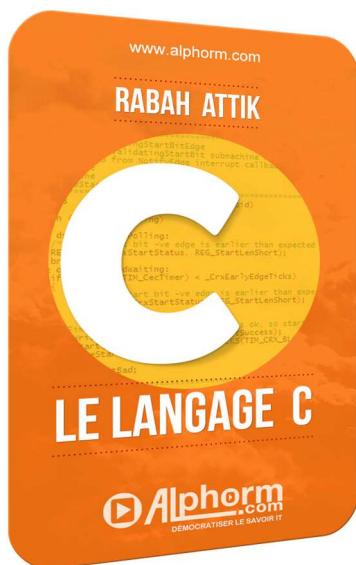
- Tests et debug
- Taille de code source
- Incompatibilité C et C++
- Intégration d'autres langage en C
- Détection de plateforme et d'architecture
- Code OS dépendant
- Frameworks

## C Ce qu'on a couvert

---

- Compilation conditionnelle par symboles
- Compilation conditionnelle par expressions
- Imbrications de directives de compilation conditionnelle
- Cas d'utilisation





**Alphorm**.com

## La bibliothèque standard Fonctions mathématiques

Site : <http://www.alphorm.com>  
Blog : <http://blog.alphorm.com>  
Forum : <http://forum.alphorm.com>

Rabah ATTIK  
Formateur et Consultant indépendant  
Ingénierie système embarqué

Formation Le langage C

alphorm.com™©

### C Plan

- Fichier <math.h>
  - Fonctions trigonométriques
  - Fonctions hyperboliques
  - Fonction exponentielle et logarithme
  - Fonction puissance
  - Autres fonctions



Formation Le langage C

alphorm.com™©

## C Fonctions trigonométriques

- **Arc cosinus:** double `acos(double x)` donne la valeur principale en radian
- **Arc sinus:** double `asin(double x)` donne la valeur principale en radian
- **Arc tangente:** double `atan(double x)` donne la valeur principale en radian
- **Arc tangente:** double `atan2(double y, double x)` donne la valeur principale en radian de  $y/x$
- **Cosinus:** double `cos(double x)`: valeur en radian
- **Sinus:** double `sin(double x)`: valeur en radian
- **Tangente:** double `tan(double x)`: valeur en radian

## C Fonctions hyperboliques

- **Cosinus hyperbolique:** double `cosh(double x)`: valeur en radian
- **Sinus hyperbolique:** double `sinh(double x)`: valeur en radian
- **Tangente hyperbolique:** double `tanh(double x)`: valeur en radian

## C Fonction exponentielle et logarithme

- Exponentielle: `double exp(double x);`
- FREXP: `double frexp(double x, int*exp);`
- LOG: `double log(double x);`
- LOG10: `double log10(double x);`
- MODF: `double modf( double x, double* adr_entier);`

## C Fonctions puissance

- Puissance: `double pow(double x, double y);`
  - X puissance Y
- Racine carré arithmétique: `double sqrt(double x);`

## C Autres fonctions

---

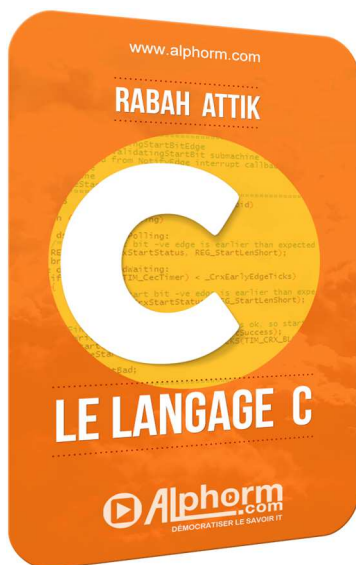
- Ceil: double ceil(double x);
- Fabs double fabs(double x);
- Floor: double floor(double x);
- Fmod: double fmod(double x, double y);

## C Ce qu'on a couvert

---

- Fichier <math.h>
  - Fonctions trigonométriques
  - Fonctions hyperboliques
  - Fonction exponentielle et logarithme
  - Fonction puissance
  - Autres fonctions





**Alphorm**.com

## La bibliothèque standard

### Entrées/sorties et fichier

Site : <http://www.alphorm.com>  
Blog : <http://blog.alphorm.com>  
Forum : <http://forum.alphorm.com>

**Rabah ATTIK**

Formateur et Consultant indépendant  
Ingénierie système embarqué

Formation Le langage C

alphorm.com™©

## C Plan

- **stdio.h : Entrées – Sorties**
  - Présentation de la bibliothèque standard d'entrées / sorties
  - Types prédéfinis
  - Constantes prédéfinies
  - Fonctions d'écriture formatée
  - Fonctions de lecture formatée
  - Fonctions d'entrées-sorties de caractères
  - Fonctions d'entrées-sorties sans formatage
  - Fonctions d'accès aux fichiers
  - Fonctions d'opérations sur les fichiers
  - Fonctions agissant sur le pointeur de fichier
  - Fonction de gestion des erreurs d'entrée-sortie



Formation Le langage C

alphorm.com™©

## C Présentation

- <assert.h> Diagnostic à l'exécution
- <ctype.h> Manipulation des caractères
- <errno.h> Gestion des erreurs
- <float.h> Limites de l'arithmétique flottante
- <limits.h> Limites de l'arithmétique entière
- <locale.h> Support multilingue
- <math.h> La bibliothèque mathématique
- <setjmp.h> Transfert d'exécution
- <signal.h> Gestion des signaux
- <stdarg.h> Gestion des arguments
- <stddef.h> Définitions générales
- **<stdio.h> Entrées/Sorties**
- <stdlib.h> Utilitaires d'usage général
- <string.h> Manipulation des chaînes de caractères
- <time.h> Manipulation des mesures de temps

## C Types prédéfinies

- Types connus :
  - int / char / float
  - struct / union
  - unsigned / signed / short / long
- Types prédéfinis :
  - size\_t
  - FILE
  - fpos\_t



## C Constantes prédéfinies

---

- NULL
- IOFBF, IOLBF, IONBF
- BUFSIZ
- EOF
- FOPEN\_MAX
- L\_tmpnam
- SEEK\_CUR, SEEK\_END, SEEK\_SET
- TMP\_MAX
- stderr
- stdin
- stdout

## C Fonctions d'accès aux fichiers

---

- FILE\* fopen(const char\* nom\_fichier, const char\* mode)
- FILE\* freopen(const char\* nom\_fichier, const char\* mode, FILE\* flux)
- int fflush(FILE\* flux)
- int fclose(FILE\* flux)

## C Fonction d'écriture formatée

- `int fprintf(FILE* flux, const char *format, ...)`
- `int printf(const char *format, ...)`
- `int sprintf(char* chaine, const char *format, ...)`
- `int vfprintf(FILE* flux, const char *format, va_list arg)`
- `int vprintf(const char *format, va_list arg)`
- `int vsprintf(char* chaine, const char *format, va_list arg)`

## C Fonction de lecture formatée

- `int fscanf(FILE* flux, const char *format, ...)`
- `int scanf(const char *format, ...)`
- `int sscanf(char* chaine, const char *format, ...)`
- `int vfscanf(FILE* flux, const char *format, va_list arg)`
- `int vscanf(const char *format, va_list arg)`
- `int vsscanf(char* chaine, const char *format, va_list arg)`

## C Fonction d'entrées sorties de caractères

- `int fgetc(FILE* flux)`
- `char* fgets(char* chaine, int n, FILE* flux)`
- `int fputc(int c, FILE* flux)`
- `int fputs(const char* chaine, FILE* flux)`
- `int getc(FILE* flux)`
- `int getchar(void)`
- `char* gets(char* chaine)`
- `int putc(int c, FILE* flux)`
- `int putchar(int c)`
- `int puts(const char* chaine)`
- `int ungetc(int c, FILE* flux)`

## C Fonction d'entrées-sorties sans formatage

- `size_t fread(void* adr, size_t taille, size_t nblocs, FILE* flux)`
- `size_t fwrite(const void* adr, size_t taille, size_t nblocs, FILE* flux)`

## C Fonctions d'opérations sur les fichiers

- `int remove(const char* nom_fichier)`
- `int rename(const char* ancien, const char *nouveau_nom)`
- `FILE*tmpfile(void)`

## C Fonctions agissant sur le pointeur sur un fichier

- `int fgetpos(FILE* flux, fpos_t* pos)`
- `int fseek(FILE* flux, long deplacement, int origine)`
- `int fsetpos(FILE* flux, const fpos_t *pos)`
- `long ftell(FILE* flux)`
- `void rewind(FILE* flux)`

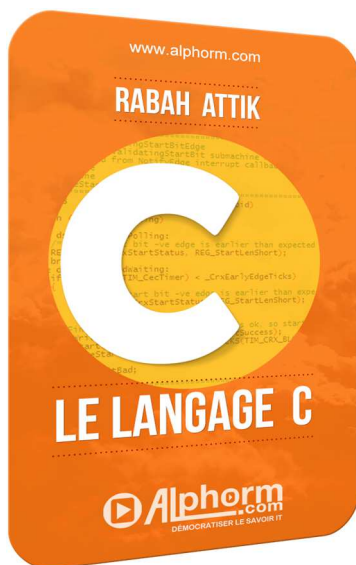
## **C** Fonction de gestion des erreurs d'entrées sorties

- void clearerr(FILE\* flux)
- int feof(FILE\* flux)
- int ferror(FILE\* flux)
- void perror(const char\* chaine)

## **C** Ce qu'on a couvert

- stdio.h: Entrées – Sorties
  - Présentation de la bibliothèques standard d'entrées / sorties
  - Types prédéfinis
  - Constantes prédéfinies
  - Fonctions d'écriture formatée
  - Fonctions de lecture formatée
  - Fonctions d'entrées-sorties de caractères
  - Fonctions d'entrées-sorties sans formatage
  - Fonctions d'accès aux fichiers
  - Fonctions d'opérations sur les fichiers
  - Fonctions agissant sur le pointeur de fichier
  - Fonction de gestion des erreurs d'entrée-sortie





**Alphorm**.com

## La bibliothèque standard

---

## Manipulation de chaînes de caractères

Site : <http://formation.inge-tech.com>  
Blog : <http://www.alphorm.com/blog>  
Forum : <http://www.alphorm.com/forum>

**Rabah ATTIK**

Formateur et consultant indépendant  
Ingénierie système embarqué  
[rabah.attik@inge-tech.com](mailto:rabah.attik@inge-tech.com)

Formation Le langage C

alphorm.com™©

### C Plan

---

- La longueur: strlen
- La copie: strcpy / strncpy
- La concaténation: strcat / strncat :
- La comparaison: strcmp :
- La recherche
  - strchr / strrchr
  - strstr
  - strpbrk
  - strspn / strcspn
- L'éclatement: strtok
- La conversion
  - strcod / strtol / strtol
  - atoi / atol / atof
- La manipulation d'octets:
  - memcpy / memmove / memcmp
  - memset / memchar

Formation Le langage C

alphorm.com™©

## C La longueur: strlen

- `size_t strlen (const char *chaine)` (string.h)
- Fournit la longueur d'une chaîne dont on lui transmet l'adresse
  - Paramètre d'entrée: **chaine**: adresse de la chaîne
  - Valeur de retour: `size_t` (*unsigned int* définie par *typedef*)

## C La copie: strcpy / strncpy

- `char *strcpy(char *dest, const char *source);`
  - **dest**: adresse à laquelle sera copiée la chaîne
  - **source**: adresse de la chaîne à copier
  - **Valeur de retour**: adresse de dest
- N'effectue pas de contrôle de longueur et ne contrôle pas la présence de '\0' en fin de chaîne
- `char *strncpy(char *dest, const char *source, size_t longueur);`
  - **dest**: adresse à laquelle sera copiée la chaîne
  - **source**: adresse de la chaîne à copier
  - **longueur**: nombre maximal de caractères copiés y compris le 0 de fin
  - **Valeur de retour**: adresse de dest

## C La concaténation: strcat / strncat

- `char *strcat (char *dest, const char *source)` (string.h)
  - Recopie la chaîne située à l'adresse source à la fin de la chaîne d'adresse dest, le `'\0'` est substituée par le premier caractère de *source*
  - Pas de contrôle de longueur, pas d'ajout `'\0'`
  - Ne gère pas le chevauchement de chaîne
- `char *strncat (char *dest, const char *source, size_t longueur)` (string.h)
  - Permet en plus de `strcat` de contrôler le nombre de caractère concaténés

## C La comparaison: strcmp

- `int *strcmp (const char *ch1, const char *ch2)` (string.h)
  - `ch1, ch2` : adresse des chaînes à comparer
  - Valeur de retour : négative/positive/nulle
- `int *strncmp (const char *ch1, const char *ch2, size_t longueur)` (string.h)
  - La comparaison se fait sur une longueur définie par longueur



## C La recherche: strchr / strrchr

- `char* strchr (const char *chaine, int c) (string.h)`
  - Recherche la première occurrence d'un caractère dans une chaîne
  - Retourne l'adresse du caractère trouvé ou NULL sinon
- `char* strrchr (const char *chaine, int c) (string.h)`
  - Fait comme « strchr » mais en commençant par la fin

## C La recherche: strstr

- `char* strstr (const char *chaine1, const char *chaine1) (string.h)`
  - Recherche la première occurrence d'une chaîne dans une chaîne
  - Retourne l'adresse de la sous chaîne trouvée ou **NULL** sinon

## C La recherche: strpbrk

- `char* strpbrk (const char *ch1, const char *ch2)` (string.h)
  - Permet de rechercher la première occurrence d'un des caractères appartenant à un ensemble
  - **ch1**: adresse de la chaîne pour effectuer la recherche
  - **ch2**: adresse de la chaîne contenant les différents caractères concernés
  - **Valeur de retour**: adresse de la première occurrence, dans la chaîne ch1 de la première occurrence de la chaîne ch2 ou NULL si rien n'est trouvé

## C strspn / strcspn

- `size_t strspn(const char*ch1, const char *ch2)` (string.h)
  - Analyse de préfixe
  - Retourne la longueur du segment initial répondant aux conditions
- `size_t strcspn(const char*ch1, const char *ch2)` (string.h)
  - Analyse de préfixe sur l'inexistence des caractère
  - Retourne la longueur du segment initial ne répondant pas aux conditions

## C L'éclatement: strtok

- `char* strtok (char *ch1, const char * delimitateur)` (string.h)
  - Éclater une chaîne en plusieurs chaînes en se basant sur un délimiteur
  - Exemple: éclater 14:25:52 en '14', '25', et '52'

## C La conversion: strtod / strtol / strtoul

- `double strtod(const char * ch1, char ** carinv)` (string.h)
  - Conversion d'une chaîne en un double
  - Valeur de retour: valeur du double ou 0 en cas d'erreur
- `long strtol(const char *ch1, char ** carinv, int base)` (stdlib.h)
  - Conversion d'une chaîne en un entier de type long
  - Valeur de retour: valeur du 'long' ou 0 en cas d'erreur
- `unsigned long strtoul(const char *ch1, char ** carinv, int base)` (stdlib.h)
  - Conversion d'une chaîne en entier de type unsigned long
  - Valeur de retour: valeur de l'unsigned long ou 0 en cas d'erreur

## C La conversion: atoi / atol / atof

- Conservées par la norme à titre de compatibilité:
  - `double atof(const char* chaine)` (stdlib.h)
  - `long atol(const char* chaine)` (stdlib.h)
  - `int atoi(const char* chaine)` (stdlib.h)

## C La manipulation d'octet: memcpy / memmove / memcmp

- Recopier la valeur d'un objet dont on connaît la taille et l'adresse:
  - `void *memcpy(void* dest, const void *src, size_t longueur)` (stdlib.h)
    - N'autorise pas le chevauchement
  - `void *memmove(void dest, const void *src, size_t longueur)` (stdlib.h)
    - Autorise pas le chevauchement
  - `int memcmp(const void zone1, const void *zone2, size_t longueur)` (stdlib.h)
    - Comparaison lexicographique
    - Utilise les mêmes règles que strcmp

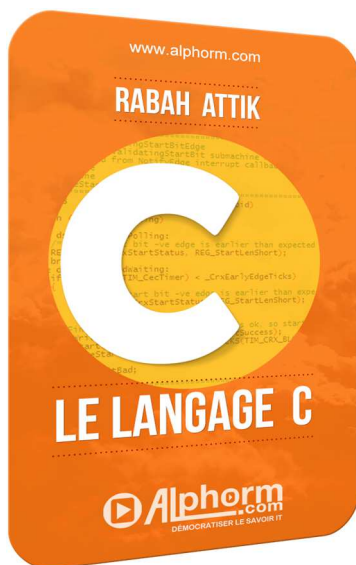
## C memset / memchr

- `void *memset (void *zone, int c, size_t longueur)` (string.h)
  - Initialise à une valeur déterminée
  - Possibilité de préciser une longueur
- `void *memchr (const void *zone, int c, size_t longueur)` (string.h)
  - Recherche la première occurrence d'une valeur donnée dans une suite d'octet

## C Ce qu'on a couvert

- Fonctions de manipulation de chaînes de caractères





# Alphorm.com

## La bibliothèque standard

### Les normes C89/C99/C11

Site : <http://formation.inge-tech.com>  
Blog : <http://www.alphorm.com/blog>  
Forum : <http://www.alphorm.com/forum>

Rabah ATTIK

Formateur et consultant indépendant  
Ingénierie système embarqué  
rabah.attik@inge-tech.com

Formation Le langage C

alphorm.com™©

## C Plan

- Les organismes de normalisation
- Historique des normes du C
- La norme ANSI C ou C89
- Les apports de la norme C99
- Les apports de la norme C11



Formation Le langage C

alphorm.com™©

## C Les organismes de normalisation

- L'ANSI (*American National Standards Institute*)
  - Organisme privé à but non lucratif qui supervise le développement de normes pour les produits, les services, les procédés, les systèmes et les employés des États-Unis.
  - [www.ansi.org](http://www.ansi.org)
  - ASCII / SCSI /
- L'ISO (*International Standard Organisation*)
  - Organisme de normalisation international composé de représentants d'organisations nationales de normalisation de 165 pays

## C Historique des normes

- C K&R puis C89 (ANSI X3.159-1989)
- C90 = C89 adopté par l'ISO (ISO/CEI 9899:1990)
- C94/C95, les correctifs
- C99 (ISO/CEI 9899:1999)
- C11 (ISO/IEC 9899:2011)

## **C** C89

---

- 10 ans d'évolution depuis la création
  - Créé en 72 et normé en 89
  - 1978: « The C programming language » décrit le langage stabilisé
  - 1983 début de la normalisation
  - Bibliothèque standard

## **C** C89

---

- Un langage complet ?
  - 1994 & 1996:
    - Bibliothèque standard, ajout de 3 fichiers
    - <http://www.open-std.org/JTC1/SC22/WG14/www/docs/tc1.htm>
    - <http://www.open-std.org/JTC1/SC22/WG14/www/docs/tc2.htm>



## C Les apports de la normes C99

---

- Les tableaux de taille variable.
- Les fonctions inline.
- Les nouveaux types de données long, \_Complex, \_Bool.
- Les pointeurs "restreints" (restrict)
- Caractères étendus

## C Les apports de la normes C99

---

- Les tableaux de taille variable  
**const int n1 = 14;**  
**int t1[n1];**
- Possibilité de passer en argument d'un fonction
- A l'intérieur du fichier (du module objet)
- Pas de tableau « static » à taille variable

## C Les apports de la normes C99

- Les fonctions inline.
  - Permet de gagner en rapidité d'exécution
  - Fonction courte appelée souvent
  - Grossi le programme

```
#define carre(a) ((a)*(a))
```

```
int main(void)
{
    int b = carre(a);
    printf("le carre de %d est %d",a, b);
    return 0;
}
```

```
inline int carre(int a) { return a*a;}
```

```
int main(void)
{
    int b = carre(a);
    printf("le carre de %d est %d",a, b);
    return 0;
}
```

## C Les apports de la normes C99

- Les nouveaux types de données long, \_Complex, \_Bool.
  - « long long »
  - « \_Complex » ( <complex.h> )
  - « \_Bool » ( <stdbool.h> )
- Les pointeurs restreints :
  - « restrict »
    - int \* restrict // pointeur restreint sur int
    - int \* restrict \* // pointeur sur pointeur restreint sur int
    - int restrict \* // ILLÉGAL
    - int \*\* restrict // pointeur restreint sur pointeur sur int

## C Les apports de la normes C99

---

- Les caractères étendus
  - En C90, la gestion des caractères spécifiques est impossible
    - wchar\_t
    - Le cast en int se fait sur wint\_t
    - Les fonctions sont reprises sous la forme wcs.... à la place de str...

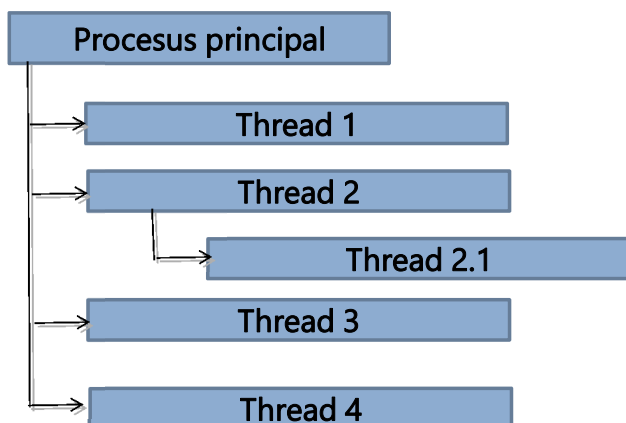
## C Les apports de la norme C11

---

- Le multithreading
- L'unicode
- Les structures et union anonymes

## C Les apports de la norme C11

- Le multithreading: (<stdatomic.h> et <threads.h>)
  - Processus « léger »



- Création et gestion de thread
- Mutex
- Variables conditionnelles
- Thread Specific storage
- Atomic Objects "atomic"

## C Les apports de la norme C11

- L'unicode :
  - UTF-8 (char)
  - UTF-16 (char16\_t)
  - UTF-32 (char32\_t)



Endian-ness / OS dependant representations

## C Les apports de la norme C11

- Les structures et unions anonymes
  - Utile pour les imbrication de structures

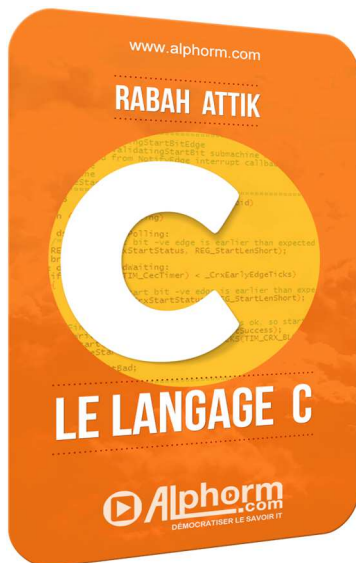
```
struct st1
{
    union
    {
        struct{int n; double d;} st2;
        struct{long q; float x;};
    };
};

struct st1 s;
s.st2.n = 1 // OK
s.n = 1 // erreur
s.q= 1 // OK
```

## C Ce qu'on a couvert

- Les organismes de normalisation
- Historique de normes en C
- C89
- C99
- C11





**Alphorm**.com

## Le langage C

---

### Conclusion

Site : <http://www.alphorm.com>  
Blog : <http://blog.alphorm.com>  
Forum : <http://forum.alphorm.com>

**Rabah ATTIK**  
Formateur et Consultant indépendant  
Ingénierie système embarqué

**Formation Le langage C**

**alphorm.com™©**

## **C** Plan

---

- Conclusion de la formation
- Présentation de la formation C Avancé



**Formation Le langage C**

**alphorm.com™©**

## **C Conclusion de la formation**

---

- Le langage C
- Chapitre 2: Premiers pas en C
- Chapitre 3: La mémoire et le programme
- Chapitre 4: Types, opérateurs et expressions
- Chapitre 5: Structures de contrôle
- Chapitre 6: Pointeurs, tableaux et chaînes de caractères
- Chapitre 7: Les types structurés
- Chapitre 8: Les fonctions
- Chapitre 9: Compilation séparée
- Chapitre 10: Le préprocesseur
- Chapitre 11: La bibliothèque standard

## **C Présentation de la formation C Avancé**

---

- C Avancé
  - Allocation dynamique
  - Les normes C99 et C11
  - Gestion avancée des fichiers
  - Gestion des signaux
  - time.h stdlib.h stdarg.h signal.h stdjmp.h stddef.h
  - Callbacks
  - Règle d'aliasing strict
  - Gestion de l'environnement
- Les caractères étendus
- Les adaptations locales
- La récursivité
- Les branchements non locaux
- Les threads
- And so on...

## C Ce qu'on a couvert

---

- Conclusion de la formation
- Présentation de la formation « C Avancé »

