

Initiation au C





Initiation au Langage C

Didier Mailliet

1/

Initiation au C

- Un premier programme
- description
- compilation
- Structure d'un programme
- Structure d'une fonction
- La fonction main
- Valeur de retour d'un programme
- Arguments d'un programme

- Le programme le plus simple que l'on puisse écrire (compiler et exécuter) est :
`main(){}`
- Ces 8 caractères seront stocké dans un fichier texte pur, par exemple : `kedal.c`
- Ne pas utiliser un traitement de texte mais un éditeur tel emacs ou dans une console linux:
`cat >kedal.c`  `main(){`  `}`  `d` 
- Les 4 lettres « `main` » doivent être collées et en minuscules
- Il peut y avoir espaces, tabulations et sauts de lignes partout ailleurs

2/

Initiation au C

- Un premier programme
- description
- compilation
- Structure d'un programme
- Structure d'une fonction
- La fonction main
- Valeur de retour d'un programme
- Arguments d'un programme

- Lorsqu'un programme est exécuté, ce sont les instructions contenues entre les `{ }` du **corps** de la fonction `main` et elles SEULES qui sont exécutées (sauf si ELLES font appel aux instructions situées ailleurs)
- <Déclarations Affectationglobales>
`main(){`
- <Déclarations Affectationlocales> ;
- <instructions> ; ← elles seules sont exécutées
- }
- <Autres Déclarations Affectationglobales>
- Les `{ }` définissent un **bloc**. Le bloc principal d'une fonction est appelé **corps**

3/

Initiation au C

- Un premier programme
- description
- Compilation 1/4
- Structure d'un programme
- Structure d'une fonction
- La fonction main
- Valeur de retour d'un programme
- Arguments d'un programme

- Pour compiler le premier programme, il suffit de taper la ligne de commande:
`gcc kedal.c`
- Il faut se trouver dans le répertoire où est sauvegardé `kedal.c` et que le chemin de `gcc` soit défini dans le path (sinon...)
- Un fichier exécutable `a.out` (`a.exe` sous dos/Win) est produit et sauvegardé dans le répertoire courant. Pour l'exécuter, il suffit de taper la ligne de commande:
`./a.out`

4/

Initiation au C

- Un premier programme
- description
- Compilation 2/4
- Structure d'un programme
- Structure d'une fonction
- La fonction main
- Valeur de retour d'un programme
- Arguments d'un programme

- `gcc` est le nom de la commande (attention à la casse)
- `kedal.c` est un paramètre de la commande (attention à la casse)
- `gcc` nécessite au moins un paramètre : le nom du fichier à compiler
- Rappel : les paramètres sont séparés les uns des autres et de la commande par un espace. Attention noms composés :
`gcc mon kedal.c` → `gcc "mon kedal.c"`
- Pour avoir une description de cette commande, taper `gcc --help` (1 paramètre) ou `man gcc`

5/

Initiation au C

- Un premier programme
- description
- Compilation 3/4
- Structure d'un programme
- Structure d'une fonction
- La fonction main
- Valeur de retour d'un programme
- Arguments d'un programme

- En pratique nous utiliserons toujours 5 paramètres (3 options et 2 arguments) :
`gcc -Wall -g -o kedal kedal.c`
- `-Wall` : permet d'afficher toutes les mises en gardes (warning)
- `-g` : permet de générer les informations pour le débogage
- `-o` : permet de choisir un nom d'exécutable qui soit différent de `a.out`
- Des Warnings apparaissent mais l'exécutable est produit
- L'exécution `./kedal` ne produit aucun résultat : le programme ne contient pas d'instruction mais ne provoque pas d'erreurs

6/

Initiation au C

- Un premier programme
- description
- Compilation 4/4
- Structure d'un programme
- Structure d'une fonction
- La fonction main
- Valeur de retour d'un programme
- Arguments d'un programme

7/

Compilation en 4 phases (-E -S -c):

- Le traitement par le préprocesseur : il effectue des transformations purement textuelles (remplacement de chaînes de caractères, inclusion d'autres fichiers, etc.)
- La compilation : le fichier engendré par le pré-processeur est traduit en assembleur
- L'assemblage : transforme le code assembleur en un fichier objet (instructions compréhensibles par le processeur)
- L'édition de liens : Une fois le code source assemblé, il faut lier entre eux les différents fichiers objets (bibliothèques,

Initiation au C

- Un premier programme
- description
- compilation
- Structure d'un programme
- Structure d'une fonction
- La fonction main
- Valeur de retour d'un programme
- Arguments d'un programme

8/

- En C, comme dans beaucoup d'autres langages, toute variable doit faire l'objet d'une déclaration avant d'être utilisée.
- Un programme C se présente de la façon suivante :

```
[directives au préprocesseur]
[Déclarations Affectation de variables externes]
[prototypes de fonctions]
[fonctions secondaires]
main() {
[Déclarations Affectation de variables internes] [instructions]
}
[autres fonctions secondaires]
```

Initiation au C

- Un premier programme
- description
- compilation
- Structure d'un programme
- Structure d'une fonction 1/4
- La fonction main
- Valeur de retour d'un programme
- Arguments d'un programme

9/

- En principe une fonction calcule un résultat qui dépend de paramètres : on dit : « le résultat est fonctions des données x, y, \dots »
- Tous les paramètres sont nécessaires et eux seuls à l'évaluation du résultat, cela ressemble aux fonctions mathématiques, par exemple : la définition $f(x, y) = x^2 + 3y$ et l'évaluation $t = f(2, 5)$ vaudrait 19
- Ci dessus x et y sont appelés paramètres **formels**, 2 et 5 paramètres **effectifs**
- L'ordre et le nombre de paramètres effectifs doivent être le même que celui des paramètres formels
 - $f(2, 5) \neq f(5, 2)$
 - $f(5)$ n'a pas de sens, pas plus que $f(5, 2, 7)$
 - $g(x) = x^2 + 3y$ et $h(x, y, z) = x^2 + 3y$
 - $g(2)$ ne permettrait pas de déterminer un résultat (il serait « fonction » de y donc le paramètre y manque)
 - $h(2, 5, 3)$ aurait la même valeur que $h(2, 5, 7)$ h n'est pas « fonction de z »

Initiation au C

- Un premier programme
- description
- compilation
- Structure d'un programme
- Structure d'une fonction 2/4
- La fonction main
- Valeur de retour d'un programme
- Arguments d'un programme

10/

- En C, on définit une fonction :

```
<type_du_résultat> <nom_de_la_fonction> (<liste_de_paramètres_typs>) {
  <déclaration_des_variables_internes>
  <liste_des_instructions>
  return <valeur_du_résultat>;
}
```
- Le `<type_du_résultat>` est l'un des types étudié plus tard. Par exemple `int` (un entier)
- Le `<liste_de_paramètres_typs>` est une répétition de zéro, une ou plusieurs fois de couples `<type> <nom_paramètre>` séparés par des virgules . Par exemple `int x, int y`
- Exemple :

```
int f (int x, int y) { return x*x+3*y ; }
```

Initiation au C

- Un premier programme
- description
- compilation
- Structure d'un programme
- Structure d'une fonction 3/4
- La fonction main
- Valeur de retour d'un programme
- Arguments d'un programme

11/

- Autres exemples :

```
int g(int t){return 3*t+2;}
int h(){return 3;}
```

 En math le graphe de la fonction g serait une droite oblique et celui de la fonction h une droite horizontale
- Remarquons que sont obligatoires :
 - Le type du résultat
 - Le nom de la fonction
 - Les parenthèses
 - Les accolades
 - Le return
 - La valeur du résultat
 - Le point-virgule

Initiation au C

- Un premier programme
- description
- compilation
- Structure d'un programme
- Structure d'une fonction 4/4
- La fonction main
- Valeur de retour d'un programme
- Arguments d'un programme

12/

- Il existe aussi des fonctions qui ne calculent pas de résultat, elles agissent sur l'environnement et se comportent comme des sous programmes. Dans d'autres langages elles sont appelées procédures
- En C `<type_du_résultat>` est `void` et ne comportent pas de `return ...;`
- On invoque le déclenchement des instructions d'une procédure ou le calcul du résultat d'une fonction dans une instruction dite appelante en mettant son nom et exactement le même nombre de paramètres formels que de paramètres effectifs.
- Ce nom de procédure est seul dans l'**instruction**
- Le nom de fonction intervient dans une **expression**

Initiation au C

- Un premier programme
- description
- compilation
- Structure d'un programme
- Structure d'une fonction
- La fonction main
- Valeur de retour d'un programme
- Arguments d'un programme

- gcc -Wall kedal.c provoque des warnings:
kedal.c:1: warning: return type defaults to 'int'
kedal.c: In function 'main':
kedal.c:1: warning: control reaches end of non-void function
- La syntaxe de la fonction main n'est pas conforme ce qui a été expliqué avant, il faudrait au moins :

```
void main(){
```
- La compilation donne cette fois:
ked.c:1: warning: return type of 'main' is not 'int'
- Pour que cela soit complètement correcte, il faut:

```
int main() { return 0;}
```


(21 caractères mini au lieu de 8 ; espaces obligatoires entre int et main puis entre return et 0)

13/

Initiation au C

- Un premier programme
- description
- compilation
- Structure d'un programme
- Structure d'une fonction
- La fonction main
- Valeur de retour d'un programme
- Arguments d'un programme

- La valeur renvoyée par le programme (résultat de la fonction main) peut être interprété par le système d'exploitation :
 - zéro lorsque le programme s'est déroulé sans erreur
 - Un entier non nul indiquant un numéro d'erreur
- Écrivons 2 programmes
 - p0.c :

```
int main() { return 0;}
```
 - p1.c :

```
int main() { return 1;}
```
- Que l'on compile

14/

Initiation au C

- Un premier programme
- description
- compilation
- Structure d'un programme
- Structure d'une fonction
- La fonction main
- Valeur de retour d'un programme
- Arguments d'un programme

- On écrit ensuite un **fichier batch** (texte pur)
 - Sous linux nommé **go** :

```
if $1 ;
then
echo pas err
else
echo err
fi
```

Sous linux, il faut lui donner le statut d'exécutable grâce à la commande:
`chmod 755 go`
 - Sous dos/win nommé **go.bat** :

```
@echo off
%1
if errorlevel 1 echo err
if not errorlevel 1 echo pas err
```
- On a donc dans le même répertoire linux:
 - go , p0 et p1 (plus peut-être p1.c et p0.c)
- et sous dos/win
 - go.bat , p0.exe et p1.exe (plus peut-être p1.c et p0.c)

15/

Initiation au C

- Un premier programme
- description
- compilation
- Structure d'un programme
- Structure d'une fonction
- La fonction main
- Valeur de retour d'un programme
- Arguments d'un programme

- On teste ensuite avec

```
./go p0
```

if \$1 ;
then
echo pas err
else
echo err
fi
- Et

```
./go p1
```
- Affichent respectivement :

```
pas err
```

```
err
```

@echo off
%1
if errorlevel 1 echo err
if not errorlevel 1 echo pas err

Quelques notions de batch sous **linux** (respectivement sous **dos**)
\$1 (respectivement **%1**) dans un fichier de commande représente le premier paramètre de la commande
Echo effectue l'affichage de ce qui suit
L'évaluation du **if** (respectivement **if errorlevel**) exécute ou non en fonction du test la suite de la commande

16/

Initiation au C

- Un premier programme
- description
- compilation
- Structure d'un programme
- Structure d'une fonction
- La fonction main
- Valeur de retour d'un programme
- Arguments d'un programme 1/2

- On a vu que :
 - Une ligne de commande était constituée par la commande (un exécutable) et ses paramètres (les options précédées par - ou -- sous linux (/ sous dos) et les arguments en général nom de fichier ou message pour echo ...)
 - Dans un batch, on peut récupérer la valeur de ces paramètres grâce à \$n (linux) %n (dos) avec n le numéro de l'argument.
- Dans un programme C, on peut récupérer les paramètres de la ligne de commande grâce aux arguments de la fonction main

17/

Initiation au C

- Un premier programme
- description
- compilation
- Structure d'un programme
- Structure d'une fonction
- La fonction main
- Valeur de retour d'un programme
- Arguments d'un programme 2/2

- Le fichier **argum.c** compilé en **argum**

```
#include <stdio.h>
int main(int n,char *param[]) {
printf("%d\n",n);
printf("%s\n",param[0]);
printf("%s\n",param[1]);
return 0;}
argum
1
argum
(null)
argum toto
2
argum
toto
```

#include <stdio.h> nécessaire pour pouvoir utiliser printf qui sert à afficher (équivalent de echo dans un batch)
char *param[] tableau de chaînes de caractères (numérotées à partir de 0)
printf("%d\n",n); affiche l'entier n
printf("%s\n",x); affiche la chaîne x
\n sert à passer à la ligne (rien à voir avec l'autre (int n)
param[i]); le ième élément du tableau
- Les commandes suivantes donnent :
argum
Le nom de la commande est dans la case 0 du tableau
n représente le nombre de mots de la ligne de commande
argum toto
Le premier paramètre de la commande est dans la case 1, le second dans la case 2 et ainsi de suite
Une case inexistante est représentée par (null)
Toutes ces notions seront détaillées par la suite

18/

Initiation au C

- Commentaires
- Identificateurs
- Directives
- define, include if - ifdef
- Type de base
- Entier, carac., boolean, flottant, pointeur, opérateurs
- Déclarations Affectation
- Portée des variables – Blocs
- Entrées / Sorties
- Chaînes de contrôle
- Structure de contrôle Sequence conditionnelles(2), boucles(3)

19/

- Ils sont ignorés du compilateur (p.p., e.l., obj.) ; c'est comme si ils n'existaient pas
- Entre `/*` et `*/`
- À partir de `//` jusqu'à la fin de ligne
- Exemple `p1.c` commenté:

```
/* Ce programme renvoie une erreur de niveau
1 au système d'exploitation */
int main(){
    return 1 ; // c'est ici le niveau 1
}
```
- Ou encore :

```
// Ce programme renvoie une erreur de niveau
// 1 au système d'exploitation
int main(){
    return 1 ; /* c'est ici le niveau 1 */
}
```

Initiation au C

- Commentaires
- Identificateurs
- Directives
- define, include if - ifdef
- Type de base
- Entier, carac., boolean, flottant, pointeur, opérateurs
- Déclarations Affectation
- Portée des variables – Blocs
- Entrées / Sorties
- Chaînes de contrôle
- Structure de contrôle Sequence conditionnelles(2), boucles(3)

20/

- Le rôle d'un identificateur est de donner un nom à une entité du programme. Plus précisément, un identificateur peut désigner :
 - un nom de variable ou de fonction,
 - un type défini par `typedef`, `struct`, `union` ou `enum`,
- Un identificateur est une suite de caractères parmi :
 - les lettres (minuscules ou majuscules, mais non accentuées),
 - les chiffres,
 - le « blanc souligné » `_`.
- Le premier caractère d'un identificateur ne peut pas être un chiffre. Éviter le `_`
- Majuscules et minuscules sont différenciées.
- Le compilateur peut tronquer les identificateurs trop longs.

Initiation au C

- Commentaires
- Identificateurs
- Directives
- define, include if - ifdef
- Type de base
- Entier, carac., boolean, flottant, pointeur, opérateurs
- Déclarations Affectation
- Portée des variables – Blocs
- Entrées / Sorties
- Chaînes de contrôle
- Structure de contrôle Sequence conditionnelles(2), boucles(3)

21/

- Comme le S.E. linux Gcc est différent de gcc
- Exemples : `var1`, `tab_23` ou `_deb` sont des identificateurs valides ; par contre, `2i` et `i:j` ne le sont pas
- Un certain nombre de mots, appelés *mots-clefs*, sont réservés pour le langage lui-même et ne peuvent pas être utilisés comme identificateurs. L'ANSI C compte 32 mots clefs (**en minuscule**) :
`auto`, `const`, `double`, `float`, `int`, `short`, `struct`, `unsigned`, `break`, `continue`, `else`, `for`, `long`, `signed`, `switch`, `void`, `case`, `default`, `enum`, `goto`, `register`, `sizeof`, `typedef`, `volatile`, `char`, `do`, `extern`, `if`, `return`, `static`, `union`, `while`

Initiation au C

- Commentaires
- Identificateurs
- Directives
- define, include if - ifdef
- Type de base
- Entier, carac., boolean, flottant, pointeur, opérateurs
- Déclarations Affectation
- Portée des variables – Blocs
- Entrées / Sorties
- Chaînes de contrôle
- Structure de contrôle Sequence conditionnelles(2), boucles(3)

22/

- Elles ne font pas partie du langage C
- Le préprocesseur est un programme exécuté lors de la première phase de la compilation. Il effectue des modifications textuelles sur le fichier source à partir de *directives*. Les différentes directives au préprocesseur, introduites par le caractère `#`, ont pour but :
 - l'incorporation de fichiers source (`#include`),
 - la définition de constantes symboliques et de macros (`#define`),
 - la compilation conditionnelle (`#if`, `#ifdef`,...).

Initiation au C

- Commentaires
- Identificateurs
- Directives
- define, include if - ifdef
- Type de base
- Entier, carac., boolean, flottant, pointeur, opérateurs
- Déclarations Affectation
- Portée des variables – Blocs
- Entrées / Sorties
- Chaînes de contrôle
- Structure de contrôle Sequence conditionnelles(2), boucles(3)

23/

- La directive `#define` permet de définir :
 - des constantes symboliques
 - `#define nom reste-de-la-ligne` demande au préprocesseur de substituer toute occurrence de *nom* par la chaîne de caractères *reste-de-la-ligne* dans la suite du fichier source. Son utilité principale est de donner un nom parlant à une constante, qui pourra être aisément modifiée. Par exemple :

```
#define NB_LIGNES 10
#define NB_COLONNES 33
#define TAILLE_MATRICE NB_LIGNES * NB_COLONNES
```
 - Il n'y a toutefois aucune contrainte sur la chaîne de caractères *reste-de-la-ligne*. On peut écrire

```
#define BEGIN {           #define END }
```
 - des macros avec paramètres. **DANGER**

```
#define CARRE(a) a * a
```

le préprocesseur remplacera `CARRE(a + b)` par `a + b * a + b` et non par `(a + b) * (a + b)`

```
#define CARRE (a) a * a
```

la chaîne de caractères `CARRE(2)` sera remplacée par `(a) a * a (2)` **espace entre CARRE et**

Initiation au C

- Commentaires
- Identificateurs
- Directives
- define, include if - ifdef
- Type de base
- Entier, carac., boolean, flottant, pointeur, opérateurs
- Déclarations Affectation
- Portée des variables – Blocs
- Entrées / Sorties
- Chaînes de contrôle
- Structure de contrôle Sequence conditionnelles(2), boucles(3)

24/

- On peut définir des constantes
 - entières (décimales, octales ou hexadécimales)
`1234`, `0243`, `0x4D`
 - Réelles (double précision par défaut)
`12.34`, `34.5e-2`
 - Caractères (entre apostrophes)
`'A'`, `'2'`, `'\"'`, `'\\'`, `'n'`, `'t'`, `'a'`
 - Chaînes de caractères (entre guillemets)
`"\"c'est\""` une `\n` chaîne`\""`
- Exemple

```
#define toto "\"c'est\""
```

```
#define tutu " une \n chaîne\""
```

```
#define titi tutu toto"xxx"
```

```
#define nombre 0243+0x4D // 163 + 77
```

Initiation au C

- Commentaires
- Identificateurs
- Directives
- define, include, if - ifdef
- Type de base
- Entier, carac., boolean, flottant, pointeur, opérateurs
- Déclarations
- Affectation
- Portée des variables - Blocs
- Entrées / Sorties
- Chaines de contrôle
- Structure de contrôle
- Sequence conditionnelles(2), boucles(3)

25/

- Elle permet d'incorporer dans le fichier source le texte figurant dans un autre fichier(recopié par le préprocesseur). Ce dernier peut être un fichier en-tête de la librairie standard (stdio.h, math.h,...) ou n'importe quel autre fichier.
- La directive #include possède deux syntaxes voisines :
 - #include <nom-de-fichier> recherche le fichier mentionné dans un ou plusieurs répertoires systèmes définis par l'implémentation (par exemple, /usr/include/);
 - #include "nom-de-fichier" recherche le fichier dans le répertoire courant (celui où se trouve le fichier source). On peut spécifier d'autres répertoires à l'aide de l'option -I du compilateur.
- La première syntaxe est généralement utilisée pour les fichiers en-tête de la librairie standard, tandis que la seconde est plutôt destinée aux fichiers créés par l'utilisateur.

Initiation au C

- Commentaires
- Identificateurs
- Directives
- define, include, if - ifdef 1/2
- Type de base
- Entier, carac., boolean, flottant, pointeur, opérateurs
- Déclarations
- Affectation
- Portée des variables - Blocs
- Entrées / Sorties
- Chaines de contrôle
- Structure de contrôle
- Sequence conditionnelles(2), boucles(3)

26/

- La compilation conditionnelle a pour but d'incorporer ou d'exclure des parties du code source dans le texte qui sera généré par le préprocesseur. Elle permet d'adapter le programme au matériel ou à l'environnement sur lequel il s'exécute, ou d'introduire dans le programme des instructions de débogage.
- Les directives de compilation conditionnelle se répartissent en deux catégories, suivant le type de condition invoquée :
 - la valeur d'une expression
 - l'existence ou l'inexistence de symboles.

Initiation au C

- Commentaires
- Identificateurs
- Directives
- define, include, if - ifdef 2/2
- Type de base
- Entier, carac., boolean, flottant, pointeur, opérateurs
- Déclarations
- Affectation
- Portée des variables - Blocs
- Entrées / Sorties
- Chaines de contrôle
- Structure de contrôle
- Sequence conditionnelles(2), boucles(3)

27/

- Sa syntaxe la plus générale est :

Le if

```
#if condition-1
partie-du-programme-1
#elif condition-2
partie-du-programme-2
...
#elif condition-n
partie-du-programme-n
#else
partie-du-programme-∞
#endif
```

Le ifdef

```
#ifndef symbole
partie-du-programme-1
#else
partie-du-programme-2
#endif
```

Si symbole est défini au moment où l'on rencontre la directive #ifndef, alors partie-du-programme-1 sera compilée et partie-du-programme-2 sera ignorée. Dans le cas contraire, c'est partie-du-programme-2 qui sera compilée. La directive #else est évidemment facultative.

De façon similaire, on peut tester la non-existence d'un symbole par : #ifndef au lieu de #ifdef
- La première condition-i vérifiée entraîne la compilation de la partie-du-programme-i correspondante et elle seule
- Si aucune condition n'est vérifiée, c'est partie-du-programme-∞ qui est compilée

Initiation au C

- Commentaires
- Identificateurs
- Directives
- define, include, if - ifdef, exemple
- Type de base
- Entier, carac., boolean, flottant, pointeur, opérateurs
- Déclarations
- Affectation
- Portée des variables - Blocs
- Entrées / Sorties
- Chaines de contrôle
- Structure de contrôle
- Sequence conditionnelles(2), boucles(3)

28/

- Un exemple :

```
#include <stdio.h>
#define TST
int main(){
    int i =
    #ifdef TST 5 #else 100 #endif ;
    printf("%d\n",i);
    return 0 ;
}
```

Selon que le symbole TST est ou non défini, le programme se comporte comme :

```
#include <stdio.h>
int main(){
    int i =5;
    printf("%d\n",i);
    return 0 ;
}

#include <stdio.h>
int main(){
    int i =100;
    printf("%d\n",i);
    return 0 ;
}
```
- En compilant ce programme, on obtient l'affichage 100
- En dé-commentant la 2^{ème} ligne et en recompilant, on obtient l'affichage 5
- Sans dé-commenter mais en compilant avec l'option D, on obtient l'affichage 5

gcc -Wall -g -DTST -o tstdef tstdef.c

Initiation au C

- Commentaires
- Identificateurs
- Directives
- define, include, if - ifdef
- Type de base
- Entier, carac., boolean, flottant, pointeur, opérateurs
- Déclarations
- Affectation
- Portée des variables - Blocs
- Entrées / Sorties
- Chaines de contrôle
- Structure de contrôle
- Sequence conditionnelles(2), boucles(3)

29/

- Le C comporte peu de types prédéfinis
 - Les entiers et caractères
 - Les nombres à virgule flottante
 - Les pointeurs (& *)qui, comme les caractères, sont des entiers particuliers
- Des types définis par le programmeur (typedef)
 - tableaux []
 - struct
 - union
 - enum

Initiation au C

- Commentaires
- Identificateurs
- Directives
- define, include, if - ifdef
- Type de base
- Entier, carac., boolean, flottant, pointeur, opérateurs
- Déclarations
- Affectation
- Portée des variables - Blocs
- Entrées / Sorties
- Chaines de contrôle
- Structure de contrôle
- Sequence conditionnelles(2), boucles(3)

30/

- Déclaration d'entier

Type	nom	Taille en bits	unsigned	signed
char	caractère	8	0..255	-128..127
short	entier court	16	0..65535	-32768.. 32767
int	entier	32	0.. 4 294 967 296	-2 147 483 648 .. 2 147 483 647
long	entier long	32	0.. 4 294 967 296	-2 147 483 648 .. 2 147 483 647
long long	entier non ansi	64	0.. 18 446 744 073 709 551 616	-9 223 372 036 854 775 808 .. 9 223 372 036 854 775 807
- Exemple:

```
unsigned long k=123456789; // déclarations
unsigned char i=123; // et initialisations
```

Initiation au C

- Commentaires
- Identificateurs
- Directives
- define, include if - ifdef
- Type de base
- Entier, carac., booléen, flottant, pointeur, opérateurs
- Déclarations Affectation
- Portée des variables - Blocs
- Entrées / Sorties
- Chaines de contrôle
- Structure de contrôle Sequence conditionnelles(2), boucles(3)

```

unsigned char b;
unsigned int c;
b=257; // warning
b=255;
b+=2; // b=b+2 pas de warning !!!
printf("%d\n",b); //affiche 1 : arithmétique modulo 256
c=b+2;
printf("%d\n",c); //affiche 257;
c=(char) (b+2); // cast
printf("%d\n",c); //affiche 1

```

31/

Initiation au C

- Commentaires
- Identificateurs
- Directives
- define, include if - ifdef
- Type de base
- Entier, carac., booléen, flottant, pointeur, opérateurs
- Déclarations Affectation
- Portée des variables - Blocs
- Entrées / Sorties
- Chaines de contrôle
- Structure de contrôle Sequence conditionnelles(2), boucles(3)

- Le fichier `<limits.h>` définit des pseudo-constantes pour les tailles des types entiers.
 - `CHAR_BIT` Nombre de bits du type char
 - `CHAR_MIN` Valeur maximale du type char
 - `CHAR_MAX` Valeur minimale du type char
 - `SHRT_MIN` Valeur minimale du type short
 - `SHRT_MAX` Valeur maximale du type short
 - `INT_MIN` Valeur minimale du type int
 - `INT_MAX` Valeur maximale du type int
 - `LONG_MIN` Valeur minimale du type long
 - `LONG_MAX` Valeur maximale du type long
 - `UCHAR_MAX` Valeur minimale du type unsigned char
 - `USHRT_MAX` Valeur maxi. du type unsigned short
 - `UINT_MAX` Valeur maximale du type unsigned int
 - `ULONG_MAX` Valeur maximale du type unsigned long
- La fonction `sizeof(x)` donne la taille en octet de la variable x

32/

Initiation au C

- Commentaires
- Identificateurs
- Directives
- define, include if - ifdef
- Type de base
- Entier, carac., booléen, flottant, pointeur, opérateurs
- Déclarations Affectation
- Portée des variables - Blocs
- Entrées / Sorties
- Chaines de contrôle
- Structure de contrôle Sequence conditionnelles(2), boucles(3)

- Il n'y a pas de type particuliers pour les caractères et le booléens
- Les booléens sont des entiers : seuls 0 et NULL (voir pointeurs) correspondent à FAUX Tout le reste vaut VRAI.
- Mais, lors d'un calcul logique VRAI a comme valeur 1
- Lorsqu'on mémorise un caractère, en réalité, on stocke son code ASCII. Ce n'est qu'une question de syntaxe:

```
char a= 'A'+ '/'12'+5; // est identique à
char a=65+32+12+5;
```
- 'A', '/65' et 65 représentent la même valeur

33/

Initiation au C

- Commentaires
- Identificateurs
- Directives
- define, include if - ifdef
- Type de base
- Entier, carac., booléen, flottant, pointeur, opérateurs
- Déclarations Affectation
- Portée des variables - Blocs
- Entrées / Sorties
- Chaines de contrôle
- Structure de contrôle Sequence conditionnelles(2), boucles(3)

- Les types float, double et long double servent à représenter des nombres en virgule flottante. Ils correspondent aux différentes précisions possibles.

float	32 bits	flottant
double	64 bits	flottant double précision
Long double	128 bits	flottant quadruple précision
- Les flottants sont généralement stockés en mémoire sous la représentation en virgule flottante normalisée. On écrit le nombre sous la forme (**Norme IEEE 754**) <http://babbar.cs.gq.edu/courses/cs341/IEEE-754.html> "signe 1,mantisse Bexpasant". En général, B=2. Le digit de poids fort de la mantisse n'est jamais nul.

34/

Initiation au C

- Commentaires
- Identificateurs
- Directives
- define, include if - ifdef
- Type de base
- Entier, carac., booléen, flottant, pointeur, opérateurs
- Déclarations Affectation
- Portée des variables - Blocs
- Entrées / Sorties
- Chaines de contrôle
- Structure de contrôle Sequence conditionnelles(2), boucles(3)

- Le fichier `<float.h>` définit des pseudo-constantes qui caractérisent les types réels.
 - `FLT_DIG` nombre de chiffres significatifs d'un float
 - `DBL_DIG` nombre de chiffres significatifs d'un double
 - `LDBL_DIG` nombre de chiffres significatifs d'un long double
 - `FLT_EPSILON` le plus petit nombre tel que $1.0 + x \neq 1.0$
 - `LDBL_EPSILON` le plus petit nombre tel que $1.0 + x \neq 1.0$
 - `FLT_MAX` le plus grand nombre du type float
 - `DBL_MAX` le plus grand nombre du type double
 - `LDBL_MAX` le plus grand nombre du type long double
 - `FLT_MIN` le plus petit nombre du type float
 - `DBL_MIN` le plus petit nombre du type double
 - `LDBL_MIN` le plus petit nombre du type long double

35/

Initiation au C

- Commentaires
- Identificateurs
- Directives
- define, include if - ifdef
- Type de base
- Entier, carac., booléen, flottant, pointeur, opérateurs
- Déclarations Affectation
- Portée des variables - Blocs
- Entrées / Sorties
- Chaines de contrôle
- Structure de contrôle Sequence conditionnelles(2), boucles(3)

- En fait, le C ne comporte que 2 types de base les entiers et les réels
- Lorsque l'on déclare une variable avec son type, le compilateur est informé de la place qu'il doit réserver en mémoire
- Un octet pour un char, 2 pour un short, 4 pour un int, etc. ...
- L'opérateur `&` donne la valeur de l'adresse d'une variable et `*` la valeur « pointée » . L'adresse est aussi un entier

```
int i,*p;
i=5;
p=&i;
printf("adresse de i %d valeur %d", (int)p,*p);
```

Cast pour éviter le warning

36/

Initiation au C

- Commentaires
- Identificateurs
- Directives
- define, include if - ifdef
- Type de base
- Entier, carac., boolean, flottant, **pointeur**, opérateurs
- Déclarations
- Affectation
- Portée des variables – Blocs
- Entrées / Sorties
- Chaines de contrôle
- Structure de contrôle Sequence conditionnelles(2), boucles(3)

```
#include <stdio.h>
int main()
{
    unsigned int a,*n;
    unsigned short *p,*q;
    unsigned char *r,*s;
    a=0x12345678;
    n=&a;
    p=(short *)n;
    q=p+1;
    r=(char*)q;
    s=r+1;

    printf("Tailles : char ->%d short->%d int ->%d long ->%d long long ->%d\n",
        sizeof(char),sizeof(short),sizeof(int),sizeof(long),sizeof(long long));

    printf("%x=%d\n",a,a);
    printf("%x=%d\n",*p,*p);
    printf("%x=%d\n",*q,*q);
    printf("%x=%d\n",*r,*r);
    printf("%x=%d\n",*s,*s);
    *r=0xAB;
    printf("%x=%d\n",a);
    printf("%x=%d\n",*(unsigned int)p);
    printf("%x=%d\n",*(unsigned int)q);
    printf("%x=%d\n",*(unsigned int)r);
    printf("%x=%d\n",*(unsigned int)s);
    return 0;
}
```

Tailles : char ->1 short->2 int ->4 long ->4 long long ->8

12345678=305419896

5678

1234

34

12

12ab5678

240ffa0

240ffa2

240ffa2

240ffa3

- Représentation de la mémoire ?
- little indian (inversion d'octets)

37/

Initiation au C

- Commentaires
- Identificateurs
- Directives
- define, include if - ifdef
- Type de base
- Entier, carac., boolean, flottant, **pointeur**, opérateurs
- Déclarations
- Affectation
- Portée des variables – Blocs
- Entrées / Sorties
- Chaines de contrôle
- Structure de contrôle Sequence conditionnelles(2), boucles(3)

```
#include <stdio.h>
int main()
{
    int x=3, *y,*z;
    y=&x;
    z=y+1;
    printf("z-y=%d\n", (int)(z-y));
    printf("z-y=%d\n", (int)z-(int)y);
    return 0;
}
```

- Affiche

z-y=1

z-y=4

- Les pointeurs sont, en général, typés
- Il est possible de définir des pointeurs non typés

```
void *p;
```

38/

Initiation au C

- Commentaires
- Identificateurs
- Directives
- define, include if - ifdef
- Type de base
- Entier, carac., boolean, flottant, **pointeur**, opérateurs
- Déclarations
- Affectation
- Portée des variables – Blocs
- Entrées / Sorties
- Chaines de contrôle
- Structure de contrôle Sequence conditionnelles(2), boucles(3)

```
...
int i=5,*p;
char *q;
void *r;
p = &i;
// q = p ; // warning: assignment from incompatible pointer type
r = p;
q = r;
...
```

- Les pointeurs typés sont incompatibles entre eux
- Les pointeurs non typés sont compatibles avec les pointeurs typés

39/

Initiation au C

- Commentaires
- Identificateurs
- Directives
- define, include if - ifdef
- Type de base
- Entier, carac., boolean, flottant, **pointeur**, opérateurs
- Déclarations
- Affectation
- Portée des variables – Blocs
- Entrées / Sorties
- Chaines de contrôle
- Structure de contrôle Sequence conditionnelles(2), boucles(3)

```
#include <stdio.h>
int main()
{
    int x=3;
    int *y;
    printf("&x=%p , *y=%p\n",&x,y);
    printf("x=%d , *y=%d\n",x,*y); // Ne pas utiliser *y non affecté
    y=&x;
    printf("x=%d , *y=%d\n",x,*y);
    *y=2;
    printf("x=%d , *y=%d\n",&x,*y);
    return 0;
}
```

- Affiche sous dos :
&x=0240FFA0 , *y=004012C6

- Affiche sous Linux :
&x=0xbffffe3d4 , *y=(nil)
Segmentation fault
Arrêt sur erreur

x=3 , *y=-1979949685

x=3 , *y=3

x=2 , *y=2

40/

Initiation au C

- Commentaires
- Identificateurs
- Directives
- define, include if - ifdef
- Type de base
- Entier, carac., boolean, flottant, **pointeur**, opérateurs
- Déclarations
- Affectation
- Portée des variables – Blocs
- Entrées / Sorties
- Chaines de contrôle
- Structure de contrôle Sequence conditionnelles(2), boucles(3)

- Utilisation normale des pointeurs :

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int *y;
    y= (int *) malloc(sizeof(int));
    *y=5;
    printf("y=%x , y=%x , *y=%d", (int)&y, (int)y, *y);
    return 0;
}
```

- Affiche :
&y=bffff2d4 , y=804a008 , *y=5
- La valeur de y dépend de l'emplacement trouvé par le système d'exploitation elle est susceptible de changer à chaque exécution.
- « géographie de la mémoire »

41/

Initiation au C

- Commentaires
- Identificateurs
- Directives
- define, include if - ifdef
- Type de base
- Entier, carac., boolean, flottant, **pointeur**, opérateurs
- Déclarations
- Affectation
- Portée des variables – Blocs
- Entrées / Sorties
- Chaines de contrôle
- Structure de contrôle Sequence conditionnelles(2), boucles(3)

- Le tableau suivant classe les opérateurs par ordres de priorité décroissants. Les opérateurs placés sur une même ligne ont même priorité. Si dans une expression figurent plusieurs opérateurs de même priorité, l'ordre d'évaluation est de gauche à droite. On préférera toutefois mettre des parenthèses en cas de doute...

```
- () [] -> .
- sizeof &(adresse) *(indirection) (type) -(unaire) ++ -- !
- * / %
- + - (binaire)
- << >>
- < <= > >=
- == !=
- &(et bit-à-bit)
- ^
- |
- &&
- ||
- : ?
- >>= <<= |= ^= &= %= /= *= -= += =
- ,
```

42/

Initiation au C

- Commentaires
- Identificateurs
- Directives
- define, include if - ifdef
- Type de base
- Entier, carac., booléen, flottant, pointeur, opérateurs
- Déclarations
- Affectation
- Portée des variables - Blocs
- Entrées / Sorties
- Chaines de contrôle
- Structure de contrôle Sequence, conditionnelles(2), boucles(3)

- Attention il existe de nombreuses similitudes
 - *(indirection) et * (multiplication)
 - (opposé) et -(soustraction)
 - &(adresse) et &(et bit-à-bit)
 - << >> et < >
 - <= >= et <= >=
 - = =
- Opérateurs spécifiques
 - ++ --
 - ,
 - << >>
 - ? :
 - = += -= *= /= %= &= ^= |=
- Opérations multiples
- Exemples :

43/


unaire / binaire

Initiation au C

- Commentaires
- Identificateurs
- Directives
- define, include if - ifdef
- Type de base
- Entier, carac., booléen, flottant, pointeur, opérateurs
- Déclarations
- Affectation
- Portée des variables - Blocs
- Entrées / Sorties
- Chaines de contrôle
- Structure de contrôle Sequence, conditionnelles(2), boucles(3)

- Chaque variable doit être déclarée pour pouvoir être utilisée
- Un pointeur doit être déclaré et alloué pour être « pleinement » utilisable
- L'affectation se fait grâce à l'opérateur = . Ne pas confondre avec la comparaison ==
- On peut déclarer une variable et l'initialiser
- Lorsque plusieurs variables sont de même type, ainsi que des pointeurs de variables de ce type, il est possible de les déclarer sur la même ligne

44/



Initiation au C

- Commentaires
- Identificateurs
- Directives
- define, include if - ifdef
- Type de base
- Entier, carac., booléen, flottant, pointeur, opérateurs
- Déclarations
- Affectation
- Portée des variables - Blocs
- Entrées / Sorties
- Chaines de contrôle
- Structure de contrôle Sequence, conditionnelles(2), boucles(3)

- Syntaxe :

```
<type> <nom> [= <valeur>]
```

```
[, <nom> [= <valeur>] ]*;
```
- Exemple

```
int a,b=5,*c,*d,**e,f=g=7;
```
- e est un pointeur sur un pointeur d'entier : Il faut 2 allocations

```
e= (int **) malloc(sizeof(int **));
*e= (int *) malloc(sizeof(int));
**e=12;
printf("e=%d , *e=%d , **e=%d\n" ,
(int)e, (int)*e,**e);
```

45/

Initiation au C

- Commentaires
- Identificateurs
- Directives
- define, include if - ifdef
- Type de base
- Entier, carac., booléen, flottant, pointeur, opérateurs
- Déclarations
- Affectation
- Portée des variables - Blocs
- Entrées / Sorties
- Chaines de contrôle
- Structure de contrôle Sequence, conditionnelles(2), boucles(3)

- Les variables globales sont déclarées et éventuellement initialisées avant

```
#include <stdio.h>
int f(int x){
    return x+1; // +y → l'utilisation de y ici provoquerait une erreur
}
int y=6; // y serait déclaré trop tard
int main(){
    int x=3,z=5;
    { int x=9;
      printf("f(%d)=%d , x=%d , z=%d\n",y,f(y),x,z);
    }
    printf("f(%d)=%d , x=%d , z=%d\n",y,f(y),x,z);
    return 0;
}
```

- Affiche :

```
f(6)=7 , x=9 , z=5
f(6)=7 , x=3 , z=5
```
- La variable utilisée est celle qui est déclarée dans le bloc le plus interne

46/

Initiation au C

- Commentaires
- Identificateurs
- Directives
- define, include if - ifdef
- Type de base
- Entier, carac., booléen, flottant, pointeur, opérateurs
- Déclarations
- Affectation
- Portée des variables - Blocs
- Entrées / Sorties
- Chaines de contrôle
- Structure de contrôle Sequence, conditionnelles(2), boucles(3)

- Utiliser #include <stdio.h>
 - printf : Elle permet l'écriture formatée sur le flux standard de sortie *stdout* (l'écran par défaut).

```
int printf( const char *format [, arg [, arg]...]);
```
 - puts : Elle permet d'écrire, à la position courante, la chaîne de caractères pointée par s sur la sortie standard.

```
int puts(const char *s);
```
 - putchar (macro) : Elle permet d'écrire, à la position courante, le caractère c sur la sortie standard.

```
int putchar(int c);
```
- Entrées de caractères
 - scanf


```
int scanf( const char *format [, pointer[,
```
 - gets

```
char *gets(char *s);
```
 - getchar (macro)

```
int getchar();
```
 - sscanf et sprintf : sont analogue à scanf (printf), sauf qu'à la place de lire l'entrée (sortie) standard, elle lit (écrit) la chaîne d'adresse s.

```
int sscanf(char *s, char *format, pointer ...);
int sprintf(char *s, char *format, arg ...);
```

47/



Initiation au C

- Commentaires
- Identificateurs
- Directives
- define, include if - ifdef
- Type de base
- Entier, carac., booléen, flottant, pointeur, opérateurs
- Déclarations
- Affectation
- Portée des variables - Blocs
- Entrées / Sorties
- Chaines de contrôle
- Structure de contrôle Sequence, conditionnelles(2), boucles(3)


- puts et gets :

```
#include <stdio.h>
int main{
    char ligne[10]; // voir tableaux
    puts("Votre texte 1 ? ");
    gets(ligne);
    return 0;
}
```
- putchar et getchar :

```
#include <stdio.h>
int main(){
    char c;
    while ((c= getchar())!=EOF) putchar(c); // voir Structure de contrôle
    return 0;
}
```

48/

```
/* résultat de l'exécution -----
Votre texte 1 ? qqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqq
Segmentation fault (core dumped)
```



Initiation au C

- Commentaires
- Identificateurs
- Directives
- define, include if - ifdef
- Type de base
- Entier, carac., boolean, flottant, pointeur, opérateurs
- Déclarations Affectation
- Portée des variables – Blocs
- Entrées / Sorties
- Chaines de contrôle
- Structure de contrôle Sequence conditionnelles(2), boucles(3)

49/

- scanf :

Ils sont introduits par le caractère % (pour-cent) et se terminent par le caractère de type de conversion suivant le format suivant :

% [largeur] [modificateur] type

- largeur** : elle précise la nombre de caractères n qui seront lus. On peut en lire moins si l'on rencontre un séparateur (espace, tabulation, retour chariot ...) ou un caractère invalide.
- modificateur** : Il précise la taille de l'objet recevant la valeur.

Modificateur	l'objet recevant est
h	un entier de type short int (d,i,o,u,x)
l	un entier de type long int (d,i,o,u,x) un réel de type double (e,f)
L	un réel de type long double (e,f,g)

Initiation au C

- Commentaires
- Identificateurs
- Directives
- define, include if - ifdef
- Type de base
- Entier, carac., boolean, flottant, pointeur, opérateurs
- Déclarations Affectation
- Portée des variables – Blocs
- Entrées / Sorties
- Chaines de contrôle
- Structure de contrôle Sequence conditionnelles(2), boucles(3)

50/

- scanf (suite) :

- type** : type de l'objet pointé par arg.

type	Type de l'objet pointé
d	signed int exprimé en décimal
o	signed int exprimé en octal
u	unsigned int exprimé en décimal
x	int (signed ou unsigned) exprimé en hexadécimal
f,e,g	réel
c	suivant la largeur : largeur non spécifiée ou égale à 1 : caractère largeur différente de 1 : une chaîne de caractères une chaîne de caractères
s	pointeur exprimé en hexadécimal
p	

- Exemple :

```
#include <stdio.h>
int main(){
    int i,j;
    double d;
    char tab[81];
    // voir tableaux
    printf("entier: ");
    scanf("%d", &i);
    printf("2 entiers et 1 double: ");
    scanf("%d%d%lf", &i, &j, &d);
    printf("chaîne (sans espace): ");
    scanf("%80s", tab); // le caractère '\0' est
                        // automatiquement ajouté
                        // à la fin de la chaîne tab
```

Initiation au C

- Commentaires
- Identificateurs
- Directives
- define, include if - ifdef
- Type de base
- Entier, carac., boolean, flottant, pointeur, opérateurs
- Déclarations Affectation
- Portée des variables – Blocs
- Entrées / Sorties
- Chaines de contrôle
- Structure de contrôle Sequence conditionnelles(2), boucles(3)

51/

- printf :

- Valeur retournée** : le nombre d'octets effectivement écrits ou la constante EOF (-1) en cas d'erreur.
- Spécificateurs de format** :

% [drapeaux] [largeur] [.precision] [modificateur] type

- drapeaux** :

Drapeaux	Signification
rien	justifié à droite et complété à gauche par des espaces
-	justifié à gauche et complété à droite par des espaces
+	les résultats commencent toujours par le signe + ou -
espace	le signe n'est affiché que pour les valeurs négatives
#	forme alternative. Si le type de conversion est : c,s,d,i,u : sans effet o : un 0 sera placé devant la valeur x, X : 0x ou 0X sera placé devant la valeur e, E, f : le point décimal sera toujours affiché g, G : même chose que e ou E, mais sans supprimer les zéros à droite

Initiation au C

- Commentaires
- Identificateurs
- Directives
- define, include if - ifdef
- Type de base
- Entier, carac., boolean, flottant, pointeur, opérateurs
- Déclarations Affectation
- Portée des variables – Blocs
- Entrées / Sorties
- Chaines de contrôle
- Structure de contrôle Sequence conditionnelles(2), boucles(3)

52/

- printf (suite) :

- largeur** : elle précise la nombre de caractères n qui seront affichés. Si la valeur à afficher dépasse la taille de la fenêtre ainsi définie, C utilise quand même la place nécessaire.

Largeur	Effet sur l'affichage
n	affiche n caractères, complété éventuellement par des espaces
0n	affiche n caractères, complété éventuellement à gauche par des 0
*	l'argument suivant de la liste fournit la largeur

- precision** : elle précise pour :
 - un entier, le nombre de chiffres à afficher
 - un réel, le nombre de chiffres de la partie décimale à afficher (avec arrondi)
 - les chaînes, le nombre maximum de caractères à afficher.

Initiation au C

- Commentaires
- Identificateurs
- Directives
- define, include if - ifdef
- Type de base
- Entier, carac., boolean, flottant, pointeur, opérateurs
- Déclarations Affectation
- Portée des variables – Blocs
- Entrées / Sorties
- Chaines de contrôle
- Structure de contrôle Sequence conditionnelles(2), boucles(3)

53/

- printf (suite) :

- precision** :

Precision	Effet sur l'affichage
Rien	précision par défaut : d,i,o,u,x : 1 chiffre e, E, f : 6 chiffres pour la partie décimale.
0	d,i,o,u,x : précision par défaut e, E, f : pas de point décimal
n	n caractères au plus
*	l'argument suivant de la liste contient la précision

- modificateur** : Il précise comment sera interprété l'argument

Modificateur	interprétation comme
h	un entier de type short (d,i,o,u,x,X)
l	un entier de type long (d,i,o,u,x,X)
L	un réel de type long double (e,E,f,g,G)

Initiation au C

- Commentaires
- Identificateurs
- Directives
- define, include if - ifdef
- Type de base
- Entier, carac., boolean, flottant, pointeur, opérateurs
- Déclarations Affectation
- Portée des variables – Blocs
- Entrées / Sorties
- Chaines de contrôle
- Structure de contrôle Sequence conditionnelles(2), boucles(3)

54/

- printf (suite) :

- type** : type de conversion de l'argument

Type	Format de la sortie
d ou i	entier décimal signé
o	entier octal non signé
u	entier décimal non signé
x	entier hexadécimal non signé
X	entier hexadécimal non signé en majuscules
f	réel de la forme [-]dddd.dddd
e	réel de la forme [-]d.ddd e [+/-]ddd
E	comme e mais l'exposant est la lettre
g	format e ou f suivant la précision
G	comme g mais l'exposant est la lettre E
c	caractère
s	affiche les caractères jusqu'au caractère nul '\0' ou jusqu'à ce que la précision soit atteinte
p	pointeur

Initiation au C

- Commentaires
- Identificateurs
- Directives
- define, include if - ifdef
- Type de base
- Entier, carac., boolean, flottant, pointeur, opérateurs
- Déclarations
- Affectation
- Portée des variables - Blocs
- Entrées / Sorties
- Chaines de contrôle
- Structure de contrôle Sequence conditionnelles(2), boucles(3)

- printf (suite) :
- Exemple d'utilisation des formats numériques :

instruction C	résultat
printf("%d\n",12345);	12345
printf("%+d\n",12345);	+12345
printf("%08d\n",12345);	12345
printf("%8.6d\n",12345);	012345
printf("%x\n",255);	ff
printf("%X\n",255);	FF
printf("%#x\n",255);	0xff
printf("%f\n",1.23456789012345);	1.234568
printf("%10.4f\n",1.23456789);	1.2346

```

printf("Bonjour\n");
printf("Nombre %d prix %ld Total %9ld\n",nbre, prix, prix * nbre);
printf("%s est facile\n", chaine);
printf("%8.2Lf\n", result);
printf("%.*Lf\n", 8, 2, result); /* equivalent a %8.2Lf */

```

55/

Initiation au C

- Commentaires
- Identificateurs
- Directives
- define, include if - ifdef
- Type de base
- Entier, carac., boolean, flottant, pointeur, opérateurs
- Déclarations
- Affectation
- Portée des variables - Blocs
- Entrées / Sorties
- Chaines de contrôle
- Structure de contrôle Sequence conditionnelles(2), boucles(3)

- Structure de contrôle :

Ce sont elles qui déterminent le déroulement d'un programme, en C, il y en a 6 :

- La séquence : les instructions se déroulent dans le bloc courant de haut en bas
- Les conditionnelles :
 - Le si : **if**
 - Le cas : **switch**
- Les boucles :
 - La boucle tant que : **while** et **for** contrairement à d'autres langages, il n'y a pas de boucle « pour » **for** est une boucle **while** déguisée
 - La boucle faire : **do ... while**

56/

Initiation au C

- Commentaires
- Identificateurs
- Directives
- define, include if - ifdef
- Type de base
- Entier, carac., boolean, flottant, pointeur, opérateurs
- Déclarations
- Affectation
- Portée des variables - Blocs
- Entrées / Sorties
- Chaines de contrôle
- Structure de contrôle Sequence conditionnelles(2), boucles(3)

- La conditionnelle si :

```

if (test)
    action1;
else
    action2 ;

```

- Remarques:
 - pas de "then"
 - mais des () autour du "test"
 - ; à la fin des DEUX instructions
- Exemple:

```

if (a > b)
    z = a;
else
    z = b;

```

57/

Initiation au C

- Commentaires
- Identificateurs
- Directives
- define, include if - ifdef
- Type de base
- Entier, carac., boolean, flottant, pointeur, opérateurs
- Déclarations
- Affectation
- Portée des variables - Blocs
- Entrées / Sorties
- Chaines de contrôle
- Structure de contrôle Sequence conditionnelles(2), boucles(3)

- Tests imbriqués

```

if (a > b && ( a < 5 || b < 10))
    z = a;
else z = b;

```

- Rappel: arrêt de l'évaluation dès que :
 - condition fausse pour: &&
 - condition vraie pour: ||
- Booléens
- pas de type booléen en C => simulation :

```

#define False 0
#define True 1
#define Boolean int
Boolean ok, fini, status;
if (ok) /*( ok != 0)*/
    status = True;
if (! fini) /*( fini == 0)*/
    status = False;

```

58/

Initiation au C

- Commentaires
- Identificateurs
- Directives
- define, include if - ifdef
- Type de base
- Entier, carac., boolean, flottant, pointeur, opérateurs
- Déclarations
- Affectation
- Portée des variables - Blocs
- Entrées / Sorties
- Chaines de contrôle
- Structure de contrôle Sequence conditionnelles(2), boucles(3)

- Actions imbriquées

```

if (n > 0)
    if (a > b)
        z = a;
    else z = b;
if (n > 0) {
    if (a > b)
        z = a;
}
else z = b;

```

- Le " else " ne se rapporte pas au même if !!!
- Règles:
 - toujours mettre des { } pour éviter les ambiguïtés
 - indenter intelligemment le code

59/

Initiation au C

- Commentaires
- Identificateurs
- Directives
- define, include if - ifdef
- Type de base
- Entier, carac., boolean, flottant, pointeur, opérateurs
- Déclarations
- Affectation
- Portée des variables - Blocs
- Entrées / Sorties
- Chaines de contrôle
- Structure de contrôle Sequence conditionnelles(2), boucles(3)

- Cas particulier: else- if cascades

```

if (test1)
    action1;
else if (test2)
    action2 ;
else if (test3)
    action3 ;
....
else action_ par- défaut ;

```

- Structure de tests en "rateau"
- Ne pas mettre des { } inutiles
- Alternative: le switch

60/

Initiation au C

- Commentaires
- Identificateurs
- Directives
- define, include, if - ifdef
- Type de base
- Entier, carac., booléen, flottant, pointeur, opérateurs
- Déclarations
- Affectation
- Portée des variables – Blocs
- Entrées / Sorties
- Chaînes de contrôle
- Structure de contrôle
- Séquence conditionnelles(2), boucles(3)

61/

- **Switch**

```
switch (expression){
    case const1:actions1;
        break;
    case const2:    actions2;
        break;
    ...
    default:actionsdef;
}
```

- const1, const2 sont des **constantes** à valeur entière
- **break** ou **return** pour sortir du switch
- (sinon on continue jusqu'à la fin du switch)
- plusieurs actions possibles séparées par ;

Initiation au C

- Commentaires
- Identificateurs
- Directives
- define, include, if - ifdef
- Type de base
- Entier, carac., booléen, flottant, pointeur, opérateurs
- Déclarations
- Affectation
- Portée des variables – Blocs
- Entrées / Sorties
- Chaînes de contrôle
- Structure de contrôle
- Séquence conditionnelles(2), boucles(3)

62/

- Boucles while et for

```
init;
while (test) {
    actions;
    chgtst ;
}
for (init; test; chgtst) {
    actions;
}
```

- *test* = expression quelconque
- *init* et *chgtst* = instructions quelconques
- test effectué **avant** d'entrer dans la boucle
- continue tant que *test* != 0
- *Règle : utiliser for pour les boucles pour*

Initiation au C

- Commentaires
- Identificateurs
- Directives
- define, include, if - ifdef
- Type de base
- Entier, carac., booléen, flottant, pointeur, opérateurs
- Déclarations
- Affectation
- Portée des variables – Blocs
- Entrées / Sorties
- Chaînes de contrôle
- Structure de contrôle
- Séquence conditionnelles(2), boucles(3)

63/