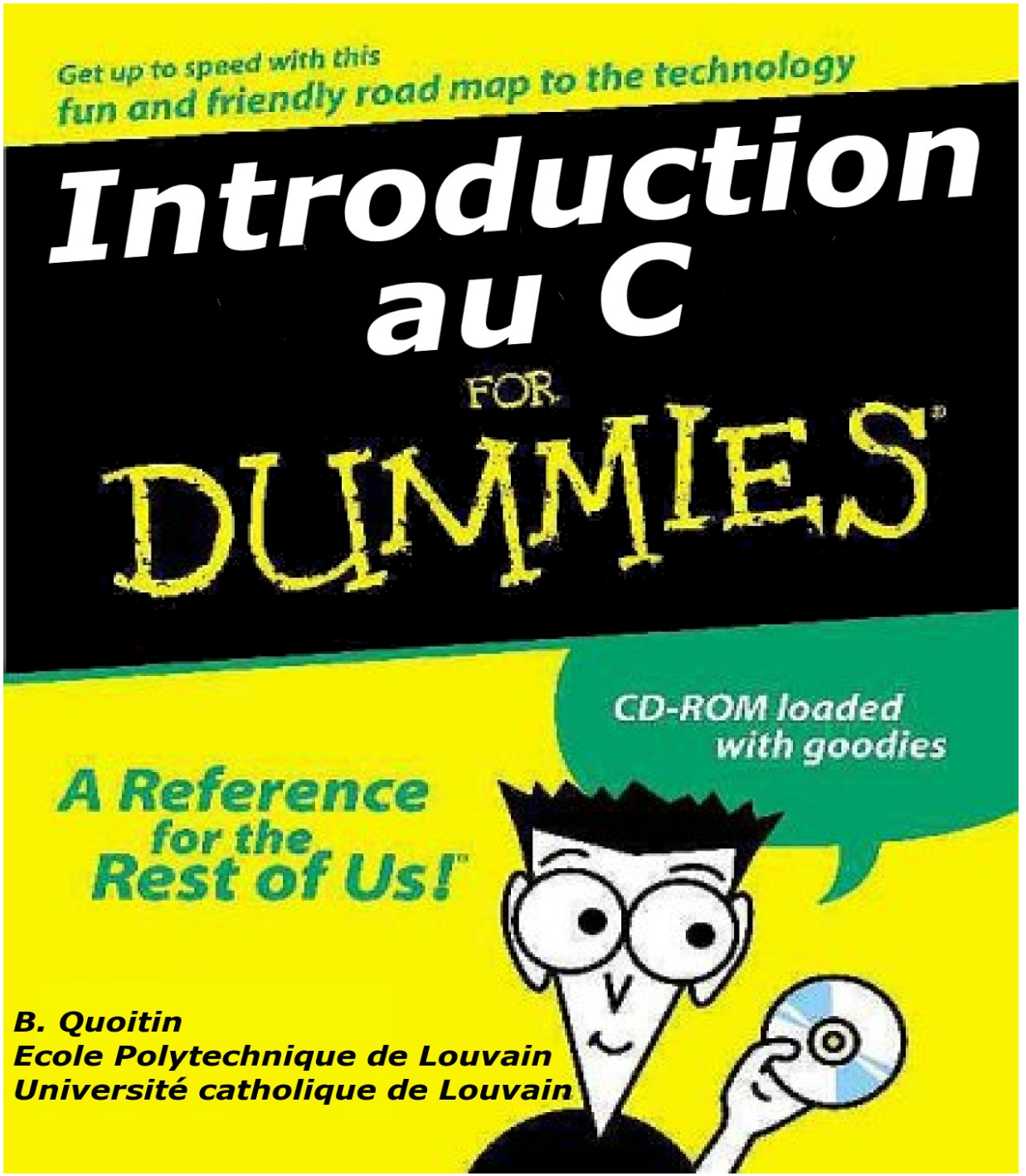
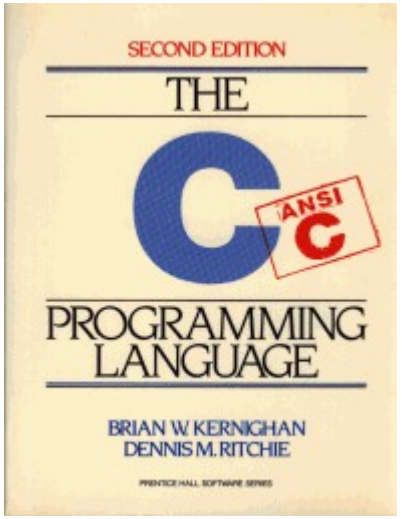

Get up to speed with this
fun and friendly road map to the technology
Introduction au C
FOR
DUMMIES
CD-ROM loaded with goodies
A Reference for the Rest of Us!
B. Quoitin
Ecole Polytechnique de Louvain
Université catholique de Louvain

Table des Matières

Ressources Utiles

Ressources Utiles



- Bouquin **« The C Programming Language »**, *Kernighan & Ritchie, 1988, Prentice Hall Inc.*
  - Disponible en bibliothèque INGI: 020-88-05
  - Version française disponible: « Le langage C ».

Ressources Utiles

- Bouquin « **C for Java programmers** », *T. Mueldner, 2000, Addison-Wesley*
  - Disponible en bibliothèque: 020-00-10

- « Learning C from Java »
  http://www.comp.lancs.ac.uk/computing/users/ss/java2c/

- « The GNU C Library » ou la « libc »
  http://www.gnu.org/software/libc/manual/html_node/index.html

Ressources Utiles

- Pages de manuel en ligne !!!

  `man 3 printf` *(doc d'une fonction)*

  `man gcc` *(doc du compilateur)*

```
PRINTF(3)                 BSD Library Functions Manual                 PRINTF(3)

NAME
     printf, fprintf, sprintf, snprintf, asprintf, vprintf, vfprintf,
     vsprintf, vsnprintf, vasprintf -- formatted output conversion

LIBRARY
     Standard C Library (libc, -lc)

SYNOPSIS
     #include <stdio.h>

     int printf(const char * restrict format, ...);
```

Exemple Introductif

Programme minimal

```c
/* My very first C program
 * B. Gates, 1980 */

#include <stdio.h>

int main()
{
  printf("MS-DOS 0.1 starting...\n");
  return 0;
}
```

Note: Tout programme C contient une fonction « main » par laquelle commence l'exécution

Compilation

- Avec GCC (GNU Compiler Collection)

```
bill$ gcc -Wall -Werror -o msdos-0.1 msdos.c
bill$ ./msdos-0.1
MS-DOS 0.1 starting...
bill$
```

`-Wall` indique que tous les avertissements du compilateur doivent être montrés.

`-Werror` indique que la compilation ne réussit que s'il n'y a ni erreur ni avertissement.

`-o <fichier>` spécifie le nom de l'exécutable à créer (ici: « msdos-0.1 »).

Les arguments suivants sont les sources.

Éléments du langage C

Constantes, Variables, Identifiants & Types

Constantes

- Constantes numériques entières
  - base décimale: **`15`**
  - base hexadécimale: **`0x0f`** (15)
  - base octale: **`017`** (15)

- Constantes numériques flottantes
  - un entier décimal: **`15`**
  - un nombre comportant un point: **`1.53`**

Constantes

- Constantes caractères
  - entre apostrophes: `'c'`
  - retour à la ligne: `'\n'`
  - tabulation: `'\t'`

- Constantes chaînes de caractères
  - entre guillemets: `"une chaîne"`

Identifiants

- Format des identifiants

```
[a-zA-Z_] [a-zA-Z_0-9]*
```

« au moins une lettre ou underscore (_) éventuellement suivis de lettres, chiffres et underscore »

- Exemples

```
x, y, z

a12345678, ma_variable, ma_fonction

12abc
```

Note: le langage C distingue les majuscules des minuscules. L'identifiant « abc » est différent de « ABC » ou « aBc »

Identifiants

- Format des identifiants

```
[a-zA-Z_] [a-zA-Z_0-9]*
```

« au moins une lettre ou underscore (_) éventuellement suivis de lettres, chiffres et underscore »

- Mots-clés

Certains identifiants bien formatés ne sont pas autorisés pour nommer des variables/fonctions/types car utilisés pas le langage C. Ce sont les mots-clés du langage. Exemples:

`break, continue, void, char, short, int, long, unsigned, signed, float, double, if, else, for, while, do, typedef, struct, return, enum, switch, ...`

Types

- Types numériques entiers

	Taille (en octets)	Domaine minimal
`char`	1	-128, 127
`unsigned char`	1	0, 255
`short`	≥ 2	-32768, 32767
`unsigned short`	≥ 2	0, 65535
`int`	≥ 2	-32768, 32767
`unsigned int`	≥ 2	0, 65535
`long`	≥ 4	-2^{31}, $2^{31}-1$
`unsigned long`	≥ 4	0, $2^{32}-1$

Note: la taille de ces types varie selon l'architecture. Par exemple, le type **`int`** a une taille de 4 octets sur un processeur 32 bits et de 8 octets sur un processeur 64 bits.

Types

- Types numériques flottants (IEEE 754)

Type	Taille (en octets)	Taille mantisse (en bits)	Taille exposant (en bits)
`float`	4	23	8
`double`	8	52	11

Représentation binaire:

S	E	M

Valeur: $(-1)^S * 2^E * (1 + M)$

où $0 \leq M < 1$, représente la partie fractionnelle

Notes:

- plus la taille de la mantisse est grande, plus la précision est grande.
- certains nombres ne sont pas représentables => « approximations ».

Variables

- Déclaration de variables

```
type identifiant;

type identifiant= valeur;

type identifiant1, identifiant2, ...;
```

- Exemples

```
int variable1;

unsigned char variable2= 'C', variable3= '\n', variable4;

float v_3;

char * v_4= "Ceci n'est pas une string";
```

Tableaux

- Tableaux

```
type identifiant [nombre_cellules] ;

identifiant[index]      avec 0 ≤ index < nombre_cellules
```

- Exemple

```
int mon_tableau[3];
mon_tableau[0]= 5;
mon_tableau[1]= 23;
mon_tableau[2]= 9;

mon_tableau[3]= 2;
mon_tableau[-1]= 2;
```

Le compilateur accepte, mais accès en dehors du tableau = danger !!!

Elements du langage C

Opérateurs

Opérateurs

- Calcul

Opérateur	Description
A **+** B	Addition de B à A
A **-** B	Soustrait B de A
A * B	Multiplication de A par B
A / B	Division de A par B
A **%** B	A modulo B (reste de la division entière de A par B)

Opérateurs

- Post/pré incrémentation/décrémentation

  `a++` : vaut a (a est aussi incrémenté)

  `++a` : vaut `a+1` (a est aussi incrémenté)

  `a--` : vaut a (a est aussi décrémenté)

  `--a` : vaut `a-1` ; (a est aussi décrémenté)

- Exemple

```
int a= 1;
int x= a++, y= ++a, z=--a;
```

Que valent x, y et z dans le code ci-dessus ?

Opérateurs

Assignation

Opérateur	Description
A= B	Affecte la valeur de B à A
A+= B	Affecte la valeur A+B à A
A-= B	Affecte la valeur A-B à A
A*= B	Affecte la valeur A*B à A
A/= B	Affecte la valeur A/B à A

Opérateurs

Priorité

Priorité	Opérateurs
1	parenthèses
2	-- ++ ! ~ -
3	* / %
4	+ -
5	< <= > >=
6	== !=
7	^
8	|
9	&& ||
10	? :
11	= += -= *= /=

Éléments du langage C

Fonctions

Fonctions

- Déclaration de fonction

```
type identifiant( [arguments] )
{
  [ déclarations de variables locales ]
  [ corps de fonction ]
  return expression;
}
```

- Exemple

```
int add( int a, int b )
{
  return a + b;
}
```

Fonctions

- Appels de fonction

```
int c= 23;
int sum= add(12, c);
```

- Ordre de déclaration
  - Si une fonction A appelle une fonction B, la fonction B doit être déclarée avant la fonction A dans le code source du programme.

Fonctions

- Portée des variables

```
int var1, var2;

int fct_B(int param) {
  int var2= var1;
  var1= param + var2;
  return var2;
}

int fct_A(int param) {
  int var1= fct_B(param) - var2;
  return var1;
}

int main() {
  var1= 10; var2= 5;
  printf("%d, %d\n", fct_A(var1), var1);
}
```

Variables « globales »

Modification de var1 par « effet de bord »

Variable « locale »

Fonctions

- Passage par valeur

```
void fonction(int param) {
  param= param * 5;
  printf("dans fonction, param=%d\n", param);
}

int main() {
  int var= 5;
  fonction(var);
  printf("dans main, param=%d\n", var);
}
```

Question: que vaut la sortie du programme ?

Fonctions

- Passage par adresse

```
void fonction(int * param) {
  (*param)= (*param) * 5;
  printf("dans fonction, param=%d\n", (*param));
}

int main() {
  int var= 5;
  fonction(&var);
  printf("dans main, param=%d\n", var);
}
```

Question: que vaut la sortie du programme ?

Note:

- « `int * param` » est une déclaration de pointeur (adresse)
- « `*param` » permet d'accéder au contenu de la variable pointée par `param`
- « `&var` » prend l'adresse de la variable `var`

Éléments du langage C

Structures de contrôle

La vérité selon C

FAUX
valeur numérique égale à 0

VRAI
valeur numérique différente de 0
(positive ou négative)

- Exemples

`0` est faux

`1` est vrai

`-1234` est vrai

Opérateurs

- Comparaison

Opérateur	Description
A **==** B	Vrai si A **égal** à B
A **!=** B	Vrai si A **différent** de B
A **<** B	Vrai si A **strictement inférieur** à B
A **>** B	Vrai si A **strictement supérieur** à B
A **<=** B	Vrai si A **inférieur ou égal** à B
A **>=** B	Vrai si A **supérieur ou égal** à B

Note: la confusion entre **=** et **==** est une erreur fréquente. Par exemple, le code suivant n'aura pas le comportement voulu...

```
if ( a = b )
  print("A (%d) est égal à B (%d) \n", a, b);
```

Question: quelle sera la sortie du programme si a vaut 2 et b vaut 3 ?

Opérateurs

- Comparaison de nombres flottants

```
float x= ...;

if ( x == 0.135742 )
   ...

if (( x > 0.135742-delta ) && ( x < 0.135742+delta ))
   ...
```

En raison de la représentation en virgule flottante, les tests d'égalité peuvent ne pas fonctionner: arrondis et/ou imprécision !

- Comparaison de chaînes

```
char user[10]= get_user();

if ( user == "sjobs" ) {
  printf("Welcome Steve !\n");
}
```

Ce test ne compare pas le contenu des chaînes de caractères, mais leur adresse en mémoire !

Opérateurs

- Logique

Opérateur	Description
A **&&** B	Vrai si A **et** B sont vrais
A **||** B	Vrai si A **ou** B sont vrais
! A	Vrai si A est faux

Note: les opérateurs logiques « **&&** » et « **||** » sont souvent confondus avec les opérateurs binaires « **&** » et « **|** ».

Structures de contrôle

- Branchement conditionnel (**if**, **if-else**)

```
if ( condition )
  bloc

if ( condition )
  bloc1
else
  bloc2
```

- Exemple

```
if ( value > max )
  printf("ERREUR: valeur trop grande !");
```

Structures de contrôle

- Branchement conditionnel (suite)

```
if ( sum > max )
  printf("ERREUR: valeur trop grande !\n");
else
  printf("La somme vaut: %d\n", sum);
```

```
if ( sum > max ) {
  sum= max;
  printf("Avertissement: valeur trop grande,\n");
  printf("                utilisation du maximum\n");
} else {
  printf("La somme vaut: %d\n", sum);
}
```

Structures de contrôle

- Branchement conditionnel (suite)

```
if ( value >= 0 )
  if ( value == 1 )
    printf("La valeur est 1\n");
else
  printf("La valeur est négative\n");
```

Ambiguïté:
on ne sait pas à quel « **if** » le « **else** » se rapporte...

```
if ( value > 0 ) {
  if ( value == 1 )
    printf("La valeur est 1\n");
} else {
  printf("La valeur n'est pas 1\n");
}
```

Structures de contrôle

- Branchements multiples (**switch**)

```
switch ( expression ) {
case value:
  instruction*
  [ break; ]
case value:
  instruction*
  [ break; ]
...
default:
  instruction*
}
```

Note: la structure **switch** fonctionne uniquement avec des types ordinaux (entiers, caractères) et pas avec des flottants ou des chaînes de caractères.

Structures de contrôle

- Exemple

```
Nombre A: 12
Nombre B: 47
Veuillez sélectionner l'opération:
  1). Addition
  2). Soustration
  3). Multiplication
  4). Division
Opération: 1
Résultat: 59
```

```
int a= get_number();
int b= get_number();
int op= get_operation();

switch ( op ) {
case 1:
  printf("Résultat: %d\n", a + b);
  break;
case 2:
  printf("Résultat: %d\n", a - b);
  break;
case 2:
  printf("Résultat: %d\n", a * b);
...
default:
  printf("ERREUR: opération non supportée !\n");
}
```

Structures de contrôle

- Exemple

```
int a= ...;

switch ( a ) {
case 1:
  printf("La variable vaut 1\n");
case 2:
  printf("La variable vaut 2\n");
...
}
```

Attention:
il est permis de ne pas mettre de « **break** »

Question: quelle est la sortie du programme si a vaut 1 ?

Structures de contrôle

- Boucle **for**

```
for ( expr-init ; expr-continuation ; expr-iteration )
  bloc
```

- Exemple: somme d'un tableau

```
int i;
int sum= 0;
char tab[16];

for ( i= 0 ; i < 16 ; i= i+1 ) {
  sum= sum + tab[i];
}
```

Structures de contrôle

- Boucle **while**

```
while ( expr-continuation )
  bloc
```

- Exemple: somme d'un tableau

```
int i= 0;
int sum= 0;
char tab[16];

while ( i < 16 ) {
  sum= sum + tab[i];
  i= i+1;
}
```

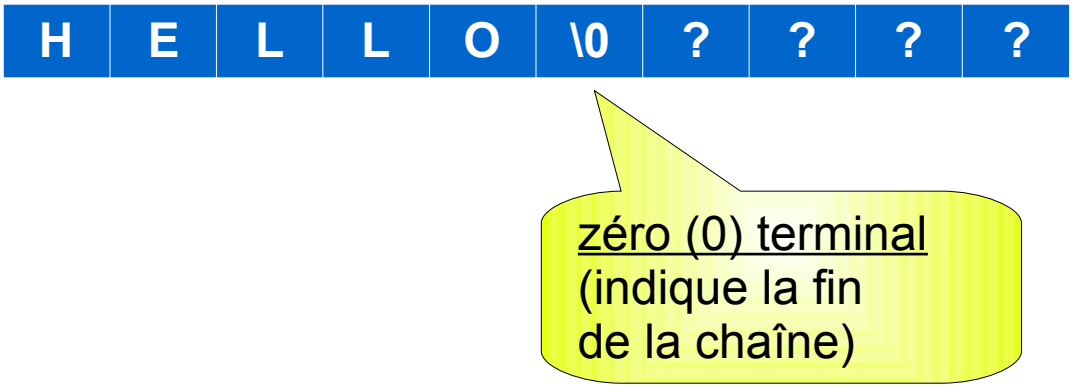
Éléments du langage C

Chaînes de caractères

Chaînes de caractères

- Représentation en mémoire

```
char tab[10]= "HELLO";
```



Notes importantes:

- la longueur de la chaîne n'est pas stockée en mémoire
- la longueur de la chaîne vaut 5
- la chaîne nécessite 6 caractères: 1 de plus pour le zéro terminal !!!
- la chaîne occupe 10 caractères en mémoire (taille du tableau)
- les 4 derniers caractères du tableau ne sont pas initialisés

Chaînes de caractères

- Tableau de caractères

```
char tab[10]= "HELLO";
```

H	E	L	L	O	\0	?	?	?	?

- Chaque caractère accessible directement

Le premier caractère est **`tab[0]`** et vaut **`'H'`**
Le second caractère est **`tab[1]`** et vaut **`'E'`**
et ainsi de suite ...
Le dernier caractère est **`tab[5]`** et vaut **`'\0'`**

Les caractères **`tab[6]`** à **`tab[9]`** sont indéfinis (non initialisés).

Chaînes de caractères

- Parcours d'une chaîne

```
char tab[10]= "HELLO";
unsigned int i= 0;

while (tab[i] != '\0') {
  printf("En position %u, caractère '%c'\n", i, tab[i]);
  i= i+1;
}
```

Chaînes de caractères

- Fonctions utiles

```
#include <string.h>

size_t strlen(const char * s);
```

Retourne le nombre de caractères de la chaîne.

```
char * strcpy(char * dst, const char * src);
```

Copie le contenu de la chaîne src dans la chaîne dst.

```
int strcmp(const char * s1, const char * s2);
```

Compare la chaîne s1 à la chaîne s2 en utilisant un ordre lexicographique.
Retourne

- **0** si s1 est égale à s2
- **<0** si s1 est plus petite que s2
- **>0** si s1 est plus grande que s2

Chaînes de caractères

- Exemple

```
char user[10]= get_user();

if (strcmp(user, "sjobs") == 0) {
  printf("Welcome Steve !\n");
}
```

Éléments du langage C

Accès aux fichiers

Accès aux fichiers

- Ouverture / fermeture de fichiers « texte »

```
#include <stdio.h>

FILE * fopen(const char * path, const char * mode);
```

Ouvre un fichier identifié par son chemin path.
Le mode d'accès peut être
- "r" pour lecture uniquement,
- "w" pour écriture

(voir man page pour autres modes).

```
int fclose(FILE * stream);
```

Ferme le fichier identifié par stream.

```
char * fgets(char * str, int size, FILE * stream);
```

Lit une ligne dans le fichier stream.
Au plus, size-1 caractères seront copiés dans la chaîne str.

Accès aux fichiers

- Exemple

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

int main() {
  char ligne[100];
  FILE * stream= fopen("mon_fichier.txt", "r");
  if (stream == NULL) {
    printf("ERREUR: impossible d'ouvrir le fichier\n");
    return EXIT_FAILURE;
  }

  if (fgets(ligne, 100, stream) != NULL)
    printf("Ligne: \"%s\"\n", ligne);

  fclose(stream);
}
```

Divers

Arguments du programme

- Arguments de la fonction « main »

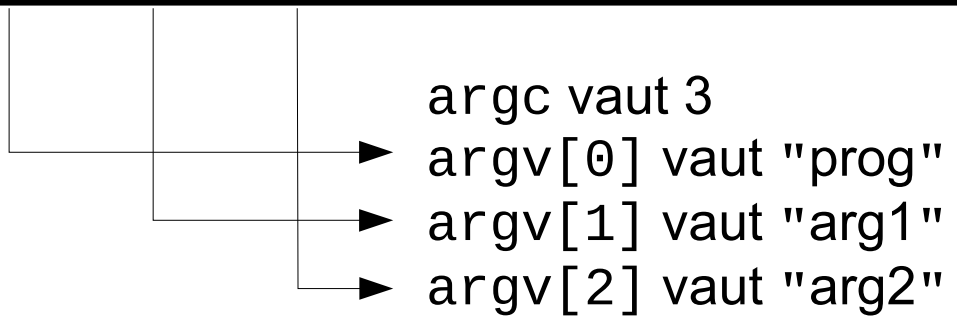
```
int main()

int main(int argc, char * argv[])
```

La fonction main peut être déclarée avec 2 arguments:
- argc indique le nombre d'arguments passés en ligne de commande
- argv est un tableau de chaînes de caractères.

- Exemple

```
toto$ ./prog arg1 arg2
```



Codes de retour

- Retour de la fonction « main »
  - passée au système. Par convention

    0 indique un fonctionnement sans erreur

    ≠ 0 indique une erreur
  - deux valeurs standard définies dans `stdlib.h`
    - `EXIT_SUCCESS` (0) et `EXIT_FAILURE` (1)

```
int main() {
  ...
  if (erreur)
    return EXIT_FAILURE;
  ...
  return EXIT_SUCCESS;
}
```

Déclaration de constantes

- Déclaration

```
#define identifiant valeur
```

- Exemples

```
#include <stdio.h>
#define TAB_LENGTH 100
#define FILE_NAME  "mon_fichier.txt"

int main() {
  double values[TAB_LENGTH];
  FILE * stream= fopen(FILE_NAME, "r");
  /* ... */
}
```

Note: il ne faut pas de « ; » après les déclarations avec **#define**

Exercices

Exercices

- Consignes
  - Une archive « `cours_c_exercices.tar.gz` » vous est fournie. Elle contient 3 répertoires nommés « `ex1` », « `ex2` » et « `ex3` ».
  - Chaque répertoire contient un fichier « `Makefile` » qui permet d'effectuer la compilation.
  - Le fichier source que vous écrivez doit avoir le même nom que le répertoire. Par exemple, dans ex1: « `ex1.c` ».
  - Pour lancer la compilation, tapez « `make` ».

Exercices

- Énoncé 1
  - Lecture et affichage d'un fichier texte
- Description
  - Lire le fichier « `mon_fichier.txt` » situé dans le répertoire courant.
  - Afficher chacune des lignes du fichier.
  - Chaque ligne sera préfixée par son numéro (en commençant à 0)

Exercices

- Énoncé 2
  - Recherche dans des fichiers
- Description
  - Argument n°1: une chaîne de caractères à rechercher.
  - Arguments n°2 et suivants: noms de fichiers (le nombre de noms n'est pas spécifié).
  - Lire chacun des fichiers.
  - Comparer chaque ligne (sans le caractère de retour) à la chaîne recherchée. Si elles sont égales, afficher le nom du fichier et le numéro de ligne.

Exercices

- Énoncé 3
  - Calcul de la moyenne d'un ensemble de nombres
- Description

  (1) Demander à l'utilisateur d'entrer un nombre en utilisant la fonction `scanf()` par exemple.

  (2) Si nombre ≥ 0, ajouter dans tableau et retourner à l'étape (1).

  (3) Si nombre < 0, afficher la moyenne des nombres du tableau et terminer le programme.

FIN

Questions ?

Remerciements:

- merci à D. Saucez pour ses commentaires sur cette 1ère version du cours.