

Initiation au C

IUP GM12 1999–2000

David Simplot

`simplot@lifl.fr`

- 4 Cours/TD (4 × 1h30) ;
 - 4 séances de travaux pratiques (4 × 2h).
- } ⇒ rapide !!!

syntaxe du C, compilation,
types de base, pointeurs, entrées/sorties, fichiers,
tableaux, structures, listes chaînées

pas d'algorithmique !!!

ex1.c : le programme C le plus simple

- En C, on écrit des fonctions :

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

type_retour nom_fonction(liste_parametres)

```
{
| code de la fonction
}
```

- Le nom de la première fonction appelée est `main`.
- Le type de retour de cette fonction est toujours `int` : un entier signé.
- Ici la fonction ne prend pas de paramètres : on note `void` ;
- La valeur de retour de la fonction est donnée par la fonction `return`. Cette fonction provoque la sortie de la fonction en cours.

- Les instructions finissent par une point-virgule.

ex2.c : Affichage d'un texte

```
int main(void)
{
    printf("J'aime le c\n");
    return(0);
}
```

- La fonction `printf` affiche sur la sortie standard la chaîne de caractères (ici donnée entre guillemets) passée en paramètre ;
- Par défaut, la sortie standard c'est le terminal dans lequel le programme a été lancé ;
- La séquence "`\n`" désigne un retour à la ligne.

Compilation d'un programme C

- On utilise la commande Unix gcc :

```
simplot@telperion-~/cunix/c] ls
ex1.c  ex2.c  ex3.c
simplot@telperion-~/cunix/c] gcc ex2.c
simplot@telperion-~/cunix/c] ls -la
total 16
drwxr-xr-x  2 simplot enseign 1024 Jan  3 19:53 .
drwxr-xr-x  3 simplot enseign 1024 Jan  3 19:54 ..
-rwxr-xr-x  1 simplot enseign 11701 Jan  3 19:53 a.out
-rw-r--r--  1 simplot enseign   32 Jan  3 19:16 ex1.c
-rw-r--r--  1 simplot enseign   59 Jan  3 19:43 ex2.c
-rw-r--r--  1 simplot enseign   79 Jan  3 20:10 ex3.c
simplot@telperion-~/cunix/c] a.out
a.out: Command not found.
```

```
simplot@telperion-~/cunix/C] echo $PATH
/usr/bin:/usr/bin:/usr/local/bin:/usr/X11R6/bin:/bin:
/usr/X11R6/bin:/home/enseign/simplot/bin:/usr/X11R6/b
in:/home/enseign/simplot/bin:/usr/X11R6/bin
simplot@telperion-~/cunix/C] setenv PATH ${PATH}:.
simplot@telperion-~/cunix/C] echo $PATH
/usr/bin:/usr/bin:/usr/local/bin:/usr/X11R6/bin:/bin:
/usr/X11R6/bin:/home/enseign/simplot/bin:/usr/X11R6/b
in:/home/enseign/simplot/bin:/usr/X11R6/bin:.
simplot@telperion-~/cunix/C] a.out
J'aime le C
simplot@telperion-~/cunix/C]
```

- **On peut spécifier à gcc le nom de l'exécutable à créer :**

```
simplot@telperion-~/cunix/C] gcc -o ex2 ex2.c
simplot@telperion-~/cunix/C] ex2
J'aime le C
```

simplot@telperion - ~/cunix/c1

- on peut demander à gcc de signaler les lignes bizarres à la compilation :

```
simplot@telperion-~/cunix/c] gcc -Wall -o ex2 ex2.c
ex2.c: In function 'main':
ex2.c:3: warning: implicit declaration of function 'p
rintf'
simplot@telperion-~/cunix/c]
```

Par défaut, le compilateur ne connaît pas la déclaration de la fonction

printf. On modifie le fichier (dans ex3.c) :

```
#include <stdio.h>

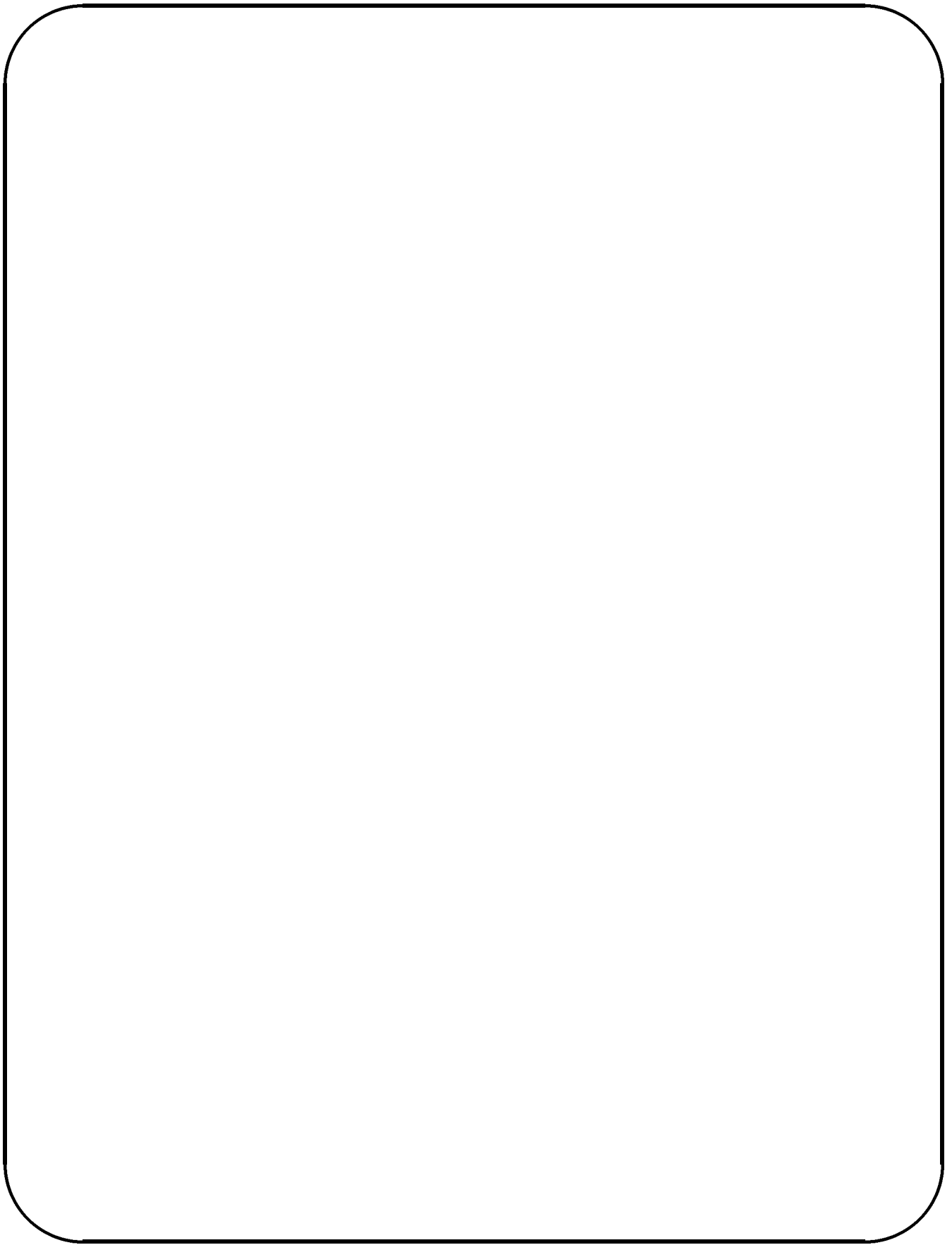
int main(void)
{
    printf("J'aime le C\n");
    return (0);
}
```

Le fichier `studio.h` contient les entêtes (déclarations) des fonctions standards d'entrées-sorties.

```
simplot@telperion-~/cunix/c] gcc -Wall -o ex3 ex3.c  
simplot@telperion-~/cunix/c] ex3  
J'aime le C  
simplot@telperion-~/cunix/c]
```

Pas de warnings !!!

C'est déjà un bon départ.



ex4. c : Saisie au clavier

```
/* Lire un entier au clavier
   et le reafficher */
#include <stdio.h>
```

```
int main(void)
{
    int n;

    printf("Valeur de n ? ");
    scanf("%d", &n);
    printf("n=%d\n", n);

    return(n);
}
```

- Le texte compris entre /* et */ est un commentaire.
- Au début d'une fonction, on déclare les variables locales de la fonction. Ici "int n" déclare une variable "n" de type int (entier signé).

- La fonction scanf permet de saisir au clavier une valeur qui sera stockée dans une variable :
 - on précise le format, ici "%d", qui spécifie que l'on lit une valeur entière,
 - on donne la (ou les variables suivantes le format) où il faut stocker les

valeurs lues...

C'est quoi ce "&" ?

- La fonction `scanf` doit modifier la valeur contenue dans la variable `n`, or

en C, tous les passages de paramètres s'effectuent par valeur.

- Comment faire pour que la fonction `scanf` puisse toute de même stocker la valeur lue au clavier dans `n` ?
- La valeur de `n` est stockée en mémoire. Pour modifier la valeur de `n`, il suffit d'écrire à l'emplacement mémoire auquel correspond `n`.

<code>n</code>	valeur de la variable <code>n</code>	type <code>int</code>
<code>&n</code>	adresse mémoire de la variable <code>n</code>	type <code>int *</code>

- En faisant scanf ("%d" , &n) on donne en paramètre (par valeur) l'adresse où il faut stocker le résultat.

ex4. c : suite des commentaires

```
/* lire un entier au clavier et le reafficher */
#include <stdio.h>

int main(void)
{
    int n;

    printf("Valeur de n ? ");
    scanf("%d", &n);
    printf("n=%d\n", n);

    return(n);
}
```

- Pour la fonction `printf` :
 - on veut afficher la valeur de `n`,
 - on précise dans le format un emplacement où il faut écrire une donnée de type entier (c'est le `%n`).
- La valeur retournée par le programme est la valeur de `n`.

```
simplot@telperion-~/cunix/c] ex4
Valeur de n ? 0
n=0
simplot@telperion-~/cunix/c] echo $status
0
simplot@telperion-~/cunix/c] ex4
Valeur de n ? 10
n=10
simplot@telperion-~/cunix/c] echo $status
10
simplot@telperion-~/cunix/c] ex4
Valeur de n ? 10
n=-1
simplot@telperion-~/cunix/c] echo $status
255
simplot@telperion-~/cunix/c]
```

ex5.c : Saisie, calcul et affichage

```
/* lire deux entiers au clavier, afficher leur somme */
#include <stdio.h>

int main(void)
{
    int a, b, s;

    printf("Valeur de a et b ? ");
    scanf("%d %d", &a, &b);
    s = a + b;
    printf("La somme de %d et %d est %d\n", a, b, s);

    return(0);
}
```

ex6.c : le tant que

```

/* calcul de factorielle */
#include <stdio.h>
int main(void)
{
    int i, n, f;

    printf("Valeur de n ? ");
    scanf("%d", &n);
    f = 1; i = 1;
    while ( i<= n )
    {
        f = f * i;
        i = i + 1;
    }
    printf ("%d!=%d\n", n, f);
    return(0);
}

```

● Syntaxe du “tant que” :

```

while (condition)
|
| bloc_instructions

```

● Un bloc d'instructions est :

- une instruction simple,
- une suite d'instructions entre acco-

lades :

```

{
| instruction1
| instruction2
}

```

ex7.c : la boucle pour

```
/* calcul de factorielle */
#include <stdio.h>

int main(void)
{
    int i, n, f;

    printf("Valeur de n ? ");
    scanf("%d", &n);
    f = 1;
    for ( i=1 ; i<=n ; i=i+1)
        f = f * i;

    printf("%d!=%d\n", n, f);

    return(0);
}
```

- Syntaxe du "pour" :

for (init; condition ; instruction)
|
bloc_instructions

- C'est équivalent à :

```
init
while (condition)
{
    bloc_instructions
    instruction;
}
```

ex 8 . c : **fonction**

```

printf("Valeur de n ? ");
scanf("%d", &n);
r = factorielle(n);
printf("%d!=%d\n", n, r);

```

```

int factorielle(int n)
{
    int i, f=1;

    for ( i=1 ; i<=n ; i++ )
        f *= i;

    return(f);
}

int main(void)
{
    int n, r;

```

- Dans la fonction `factorielle`, on déclare deux variables locales : `i` et `f`. La variable `f` est initialisée à 1.
 - "`f *= i`" est équivalent à "`f = f * i`"
 - "`i++`" est équivalent à "`i = i + 1`" et à "`i += 1`"...

e X 9 . c : **variante**

```

printf("Valeur de n ? ");
scanf("%d", &n);
printf("%d!=%d\n", n,
      factorielle(n));

int factorielle(int n)
{
    int i=1, f=1;

    while ( i <= n )
        f *= i++;

    return(f);
}

int main(void)
{
    int n;

```

- l'instruction "i++" évalue la variable i puis incrémente de un la valeur de i,
- a contrario, l'instruction "++i" incrémente i puis évalue i,
- le retour à la ligne pour l'appel à `factorielle` ne compte pas comme fin d'instruction...

ex10.c : instruction si

- Syntaxe du "si" :

```
if (condition)
|
| bloc_instructions
```

- "si" avec "sinon" :

```
if (condition)
|
| bloc_instructions
else
|
| bloc_instructions
```

```
int factorielle(int n)
{
    if ( n <= 1 )
        return(n);
    else
        return ( n*factorielle(n-1) );
}
```

- attention, le `else` se rapporte au dernier `if` ouvert...

ex11.c : variante

```
int factorielle(int n)
{
    if ( n <= 1 )
        return(n);
    return ( n*factorielle(n-1) );
}
```

- l'instruction `return` provoque la fin de la fonction en cours...
- si la condition est vérifiée, le second `return` n'est jamais exécuté.

ex12.c : procédure

```
/* calcul de factorielle */
#include <stdio.h>

void factorielle(int n, int *F)
{
    int i;

    *F=1;
    for ( i=1 ; i<=n ; i++ )
        *F = *F * i;
}

int main(void)
{
    int n, r;

    printf("Valeur de n ? ");
    scanf ("%d", &n);
    factorielle(n, &r);
    printf ("%d!=%d\n", n, r);
    return (0);
}
```

- la fonction `factorielle` est une procédure (type de retour `void`),
- elle doit stocker la factorielle de son premier paramètre `n` dans son second paramètre `F...`

ex 12 . c : procédure... suite

```

/* calcul de factorielle */
#include <stdio.h>

void factorielle(int n, int *F)
{
    int i;

    *F=1;
    for ( i=1 ; i<=n ; i++ )
        *F = *F * i;
}

int main(void)
{
    int n, r;

    printf("Valeur de n ? ");
    scanf ("%d", &n);
    factorielle(n, &r);
    printf ("%d!=%d\n", n, r);
    return (0);
}

```

- le paramètre `F` est une adresse d'un entier. On dit que c'est un pointeur vers un entier (type `int *`),
- pour accéder au contenu de la mémoire à l'endroit pointé par `F`, on utilise `*F`.

ex12. c : procédure... suite

```
void factorielle(int n, int *f)
{
    int i;
    *f=1;
    for ( i=1 ; i<=n ; i++)
        *f = *f * i;
}
```

```
int main(void)
{
    int n, r;
    ...
    factorielle(n, &r);
    ...
}
```

r	valeur de la variable r	type int
&r	adresse mémoire de la variable r	type int *
f	adresse mémoire d'un entier	type int *
*f	entier pointé par f	type int