

# Developpement logiciel en C

## Software development in C

```
int main()
{
    int variable=5;
    printf("%d\n",variable);
}

void parse_ligne_commande(const char* ligne_commande,action_a_realiser* action_demandee)
{
    assert(action_demandee!=NULL);

    // Initialise action
    // Si ligne de commande commence par '#' => ne fait rien (commentaires)
    // Si ligne de commande vaut "fin" ou "FIN" => demande action de type "fin"
    // Si ligne de commande commence par 'i' => initialise l'echiquier
    // Si ligne de commande commence par 'd'
    //   Si la syntaxe correspond a "d x0 y0 -> x1 y1"
    //     Alors demande action deplacement de (x0,y0) vers (x1,y1)
    // Si ligne de commande commence par "lit"
    //   Si la syntaxe correspond a "lit nom_fichier"
    //     Alors demande action lecture fichier NOM_FICHER
    // Si ligne de commande commence par "ecrit"
    //   Si la syntaxe correspond a "ecrit nom_fichier"
    //     Alors demande action ecriture fichier NOM_FICHER
    // Si aucune reconnaissance, affiche que la ligne est invalide

    action_initialise(action_demandee);

    //protection en mode release
    if(ligne_commande==NULL)
        return ;

    //commentaires
    if(strlen(ligne_commande)>=1 && ligne_commande[0]!='#')
        return ;

    //fin du jeu
    else if(strlen(ligne_commande)>=3 &&
            (strncmp(ligne_commande,"FIN",3)==0 || strcmp(ligne_commande,"fin",3)==0) )
    {
        action_demandee->type=fin;
        return ;
    }

    //demande de deplacement
    else if(strlen(ligne_commande)>=2 && ligne_commande[0]=='d' && ligne_commande[1]==' ')
```



# Developpement logiciel en C

6 Cours (2h)

damien.rohmer@cpe.fr

6 séances projets (4h)

damien.rohmer@cpe.fr  
martine.breda@cpe.fr  
nathael.pajani@cpe.fr  
mohamed.sallami@cpe.fr

---

(36h)

6 travaux autonomie (3h)

+

18h révision

---

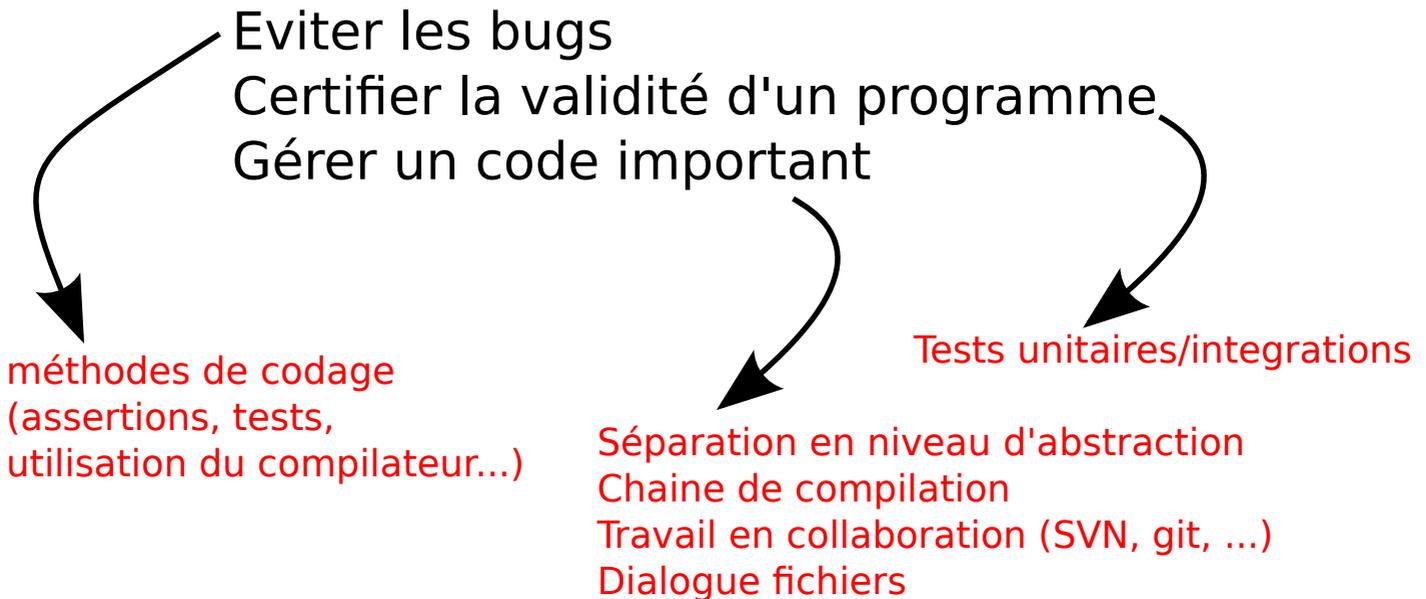
(72h)

2 notes TA  
1 note rendu projet  
1 contrôle écrit

# But du module

Connaitre/Mettre en oeuvre:

## Bonnes pratiques de codage



# Acquis du module

## Acquérir une culture générale informatique

*(implicite)*

langage du monde informatique  
licences logiciels  
cycles de développement  
logiciels de debugs, version  
...

## Devenir indépendant face à la machine

*(implicite)*

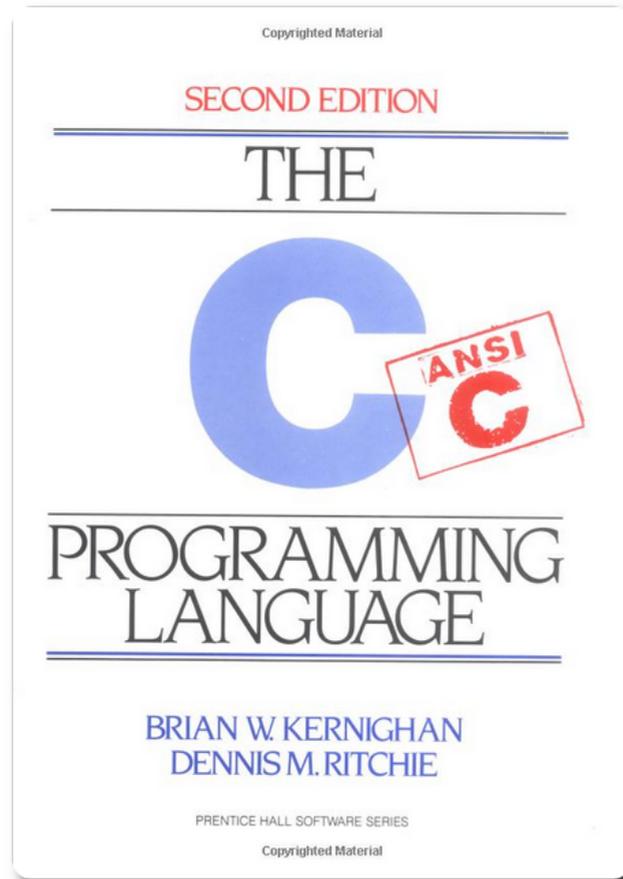
utiliser le compilateur et lire ses sorties  
debugger  
utiliser la ligne de commande sous linux  
réaliser des scripts minimalistes d'automatisation  
utiliser les Makefile  
utiliser des bibliothèques  
...

## Savoir lire et s'habituer au code de vrais projets

*(explicite)*

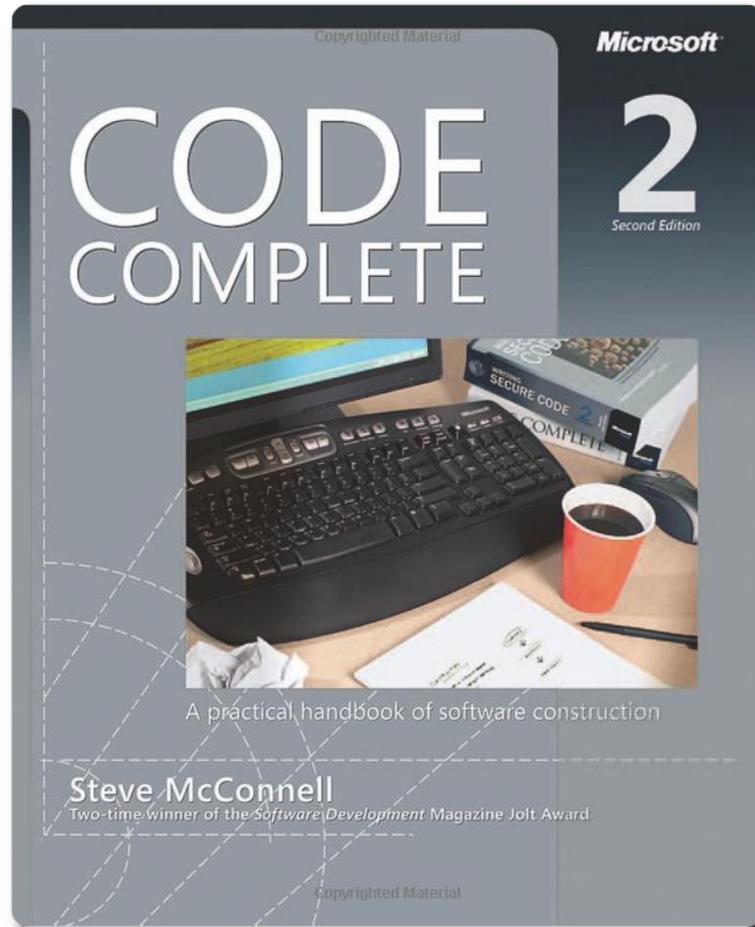
precompilateurs  
contraintes des gros codes  
méthodologie standard de développement  
connaître et reconnaître les bonnes pratiques  
...

# Bibliographie supplémentaire



Syntaxe C complète

# Bibliographie supplémentaire



Bonnes pratiques de codage (générique)

# Bibliographie supplémentaire

Les sites web:

Wikipedia

<http://www.wikipedia.org/>



Developpez.com

<http://www.developpez.com/>



Le Site du Zéro

<http://www.siteduzero.com/>



Servez-vous en pour pratiquer, progresser  
Ne le cachez pas!

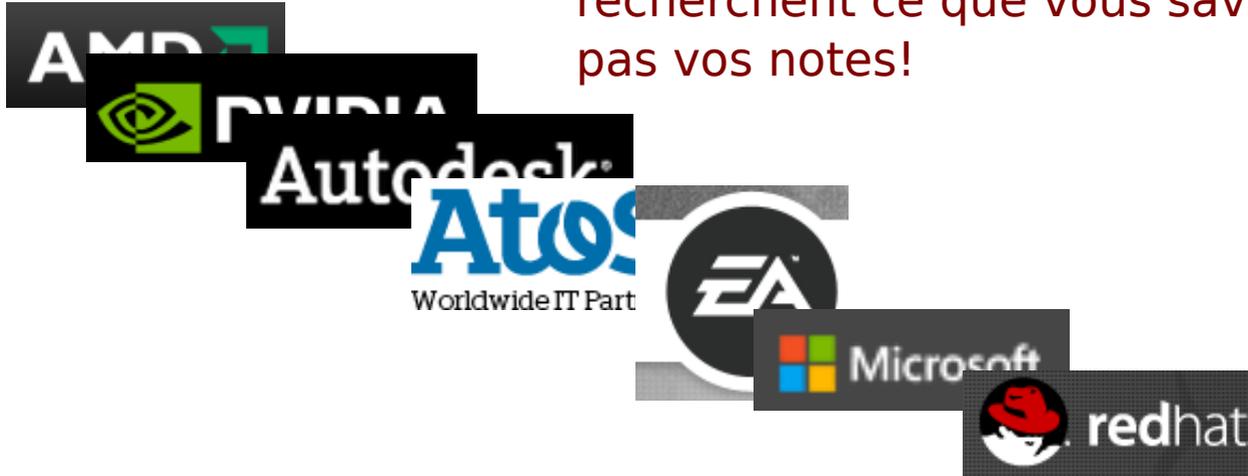
# Pratiquez

Informatique = pratique  
= expérience

**Pratiquez !**

+ Pratique => meilleur emploi

recherchent ce que vous savez faire!  
pas vos notes!



# Pratiquez

Informatique = pratique  
= expérience

**Pratiquez !**

## Outils à votre disposition:

PC Personnel +  = confort

Pensez aux solutions pas chères  
ex. leboncoin:



120 euros



180 euros



85 euros

+  
Salles infos CPE  
ouvertes jusqu'à  
**21h !**

# Sommaire

1: Introduction  
2: Présentation du projet  
3: Rappels de C  
4: Gestion de code important  
5: Licences logiciels  
6: Qualité du code

11: Mémoire dynamique  
12: Chaîne de compilation

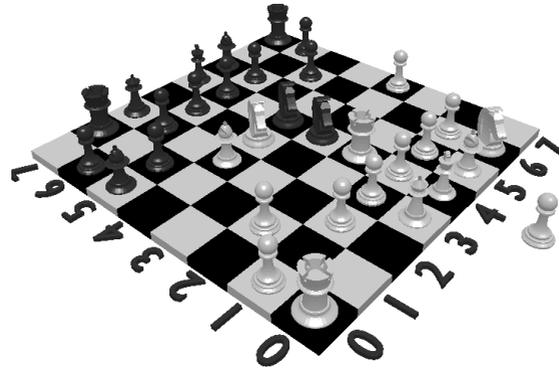
13: Entrées/sorties  
14: Gestion des erreurs  
15: Méthode de debug

7: Méthodologies de conceptions  
8: Organisation des données  
(structs/pointeurs)

16: Tests  
17: Design d'API

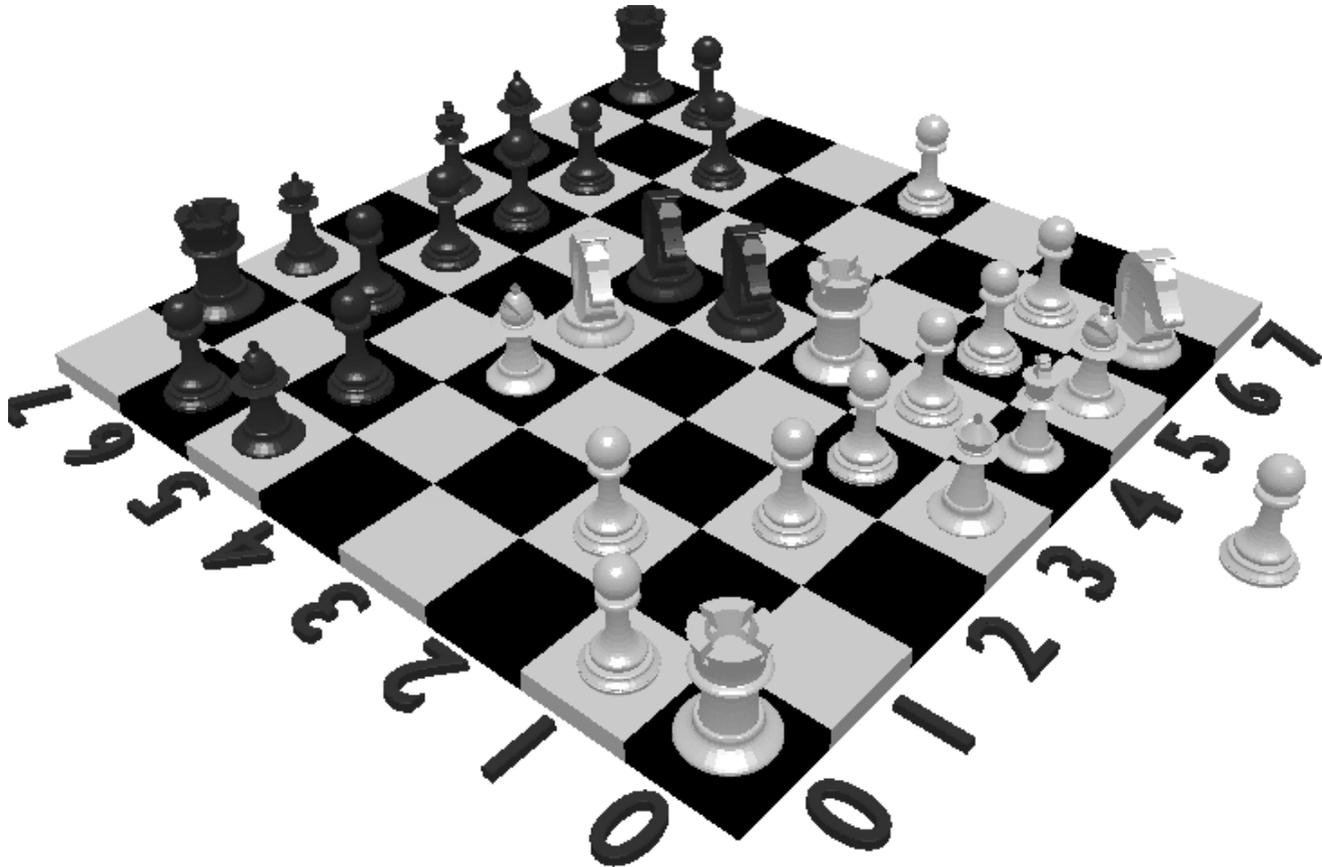
9: Code de haut niveau:  
Modélisation d'abstraction  
10: Bonnes pratiques de codage

# Le projet



# Le projet

Réaliser un jeu d'echec



# Le projet: fonctionnement

```
Terminal - damien@garonne: ~/work/2012_2013_teaching/2012_3eti_pi + - o x
File Edit View Terminal Go Help

/projet-build$ ./
CMakeFiles/ projet_3eti
[damien@garonne ~/work/2012_2013_teaching/2012_3eti_projet_c/projet_corre]
/projet-build$ ./projet_3eti

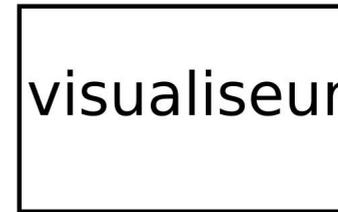
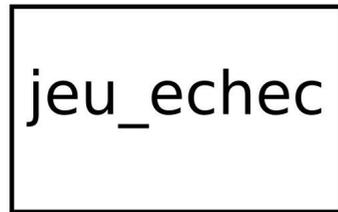
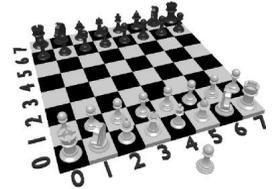
*****
Jeu echec mode interactif:
*****
Commandes:
v #           : Commentaires, tout ce qui suit # n'est pas analyse.
v i           : Initialisation du jeu d'echec.
v d x0 y0 -> x1 y1 : deplacement piece de (x0,y0) vers (x1,y1).
v lit NOM_FICHER  : Demande le chargement de l'etat de l'echiquier cont
le fichier
v ecrit NOM_FICHER : Demande l'ecriture de l'etat de l'echiquier dans le
fichier
*****

v i
Initialisation echiquier
v d 1 1 -> 1 2
J.0 deplace (1,1) => (1,2)
v d 6 6 -> 6 5
J.1 deplace (6,6) => (6,5)
v d 6 0 -> 5 2
J.0 deplace (6,0) => (5,2)
v d 6 7 -> 5 5
J.1 deplace (6,7) => (5,5)
v d 5 2 -> 5 4
Deplacement impossible
Code erreur 1
v d 5 2 -> 6 4
J.0 deplace (5,2) => (6,4)
v d 5 7 -> 7 5
J.1 deplace (5,7) => (7,5)
v d 3 1 -> 3 2
J.0 deplace (3,1) => (3,2)
v d 1 6 -> 1 5
J.1 deplace (1,6) => (1,5)
v d 2 0 -> 0 2
J.0 deplace (2,0) => (0,2)
v
```



# Le projet: coeur/graphique

```
typedef struct  
{  
    type_de_piece type;  
    int coord_x;  
    int coord_y;  
}piece;
```



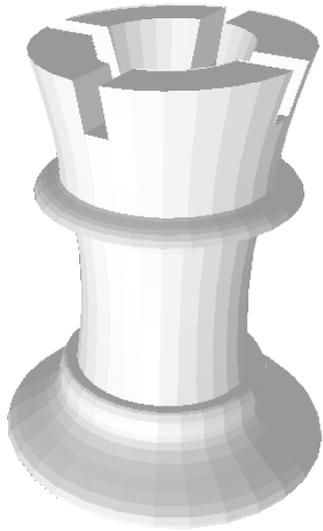
écrit



echiquier.txt

lit

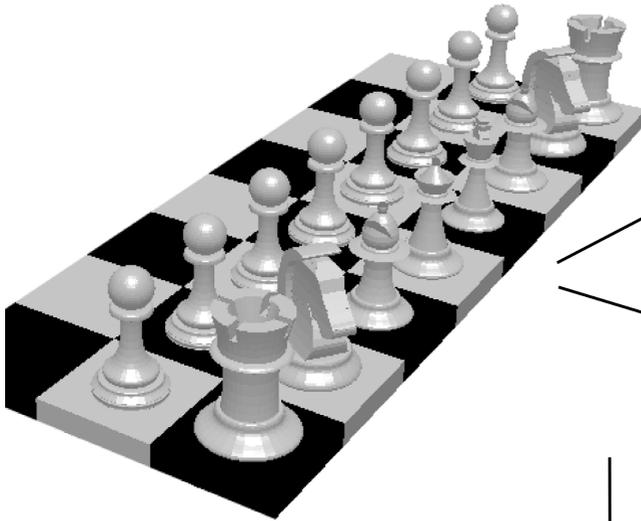
# Le projet: Piece



```
typedef struct  
{  
    type_de_piece type;  
    int coord_x;  
    int coord_y;  
}piece;
```

```
typedef enum {inconnue=0,  
             tour=1,  
             cavalier=2,  
             fou=3,  
             reine=4,  
             roi=5,  
             pion_blanc=6,  
             pion_noir=7} type_de_piece;
```

# Le projet: Jeu

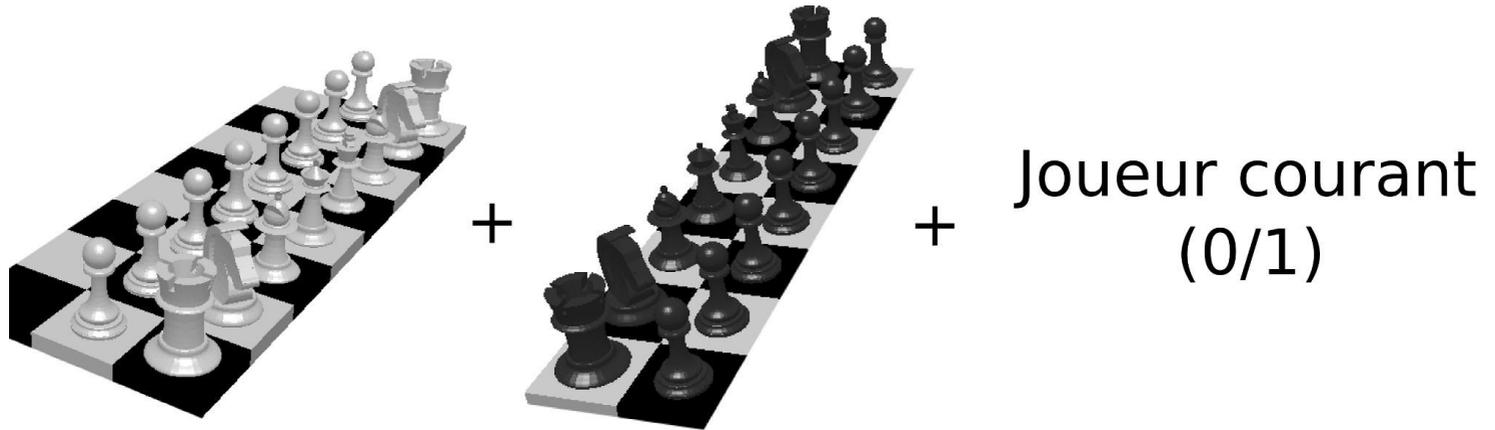


```
typedef struct{  
    piece ensemble_de_piece[MAX_PIECE];  
}jeu;
```

16

```
ensemble_de_piece[0]= ♟ (0,1)  
    ...  
ensemble_de_piece[7]= ♟ (0,7)  
ensemble_de_piece[8]= ♖ (0,0)  
ensemble_de_piece[9]= ♗ (7,0)  
ensemble_de_piece[10]= ♘ (1,0)  
    ...
```

# Le projet: Echiquier

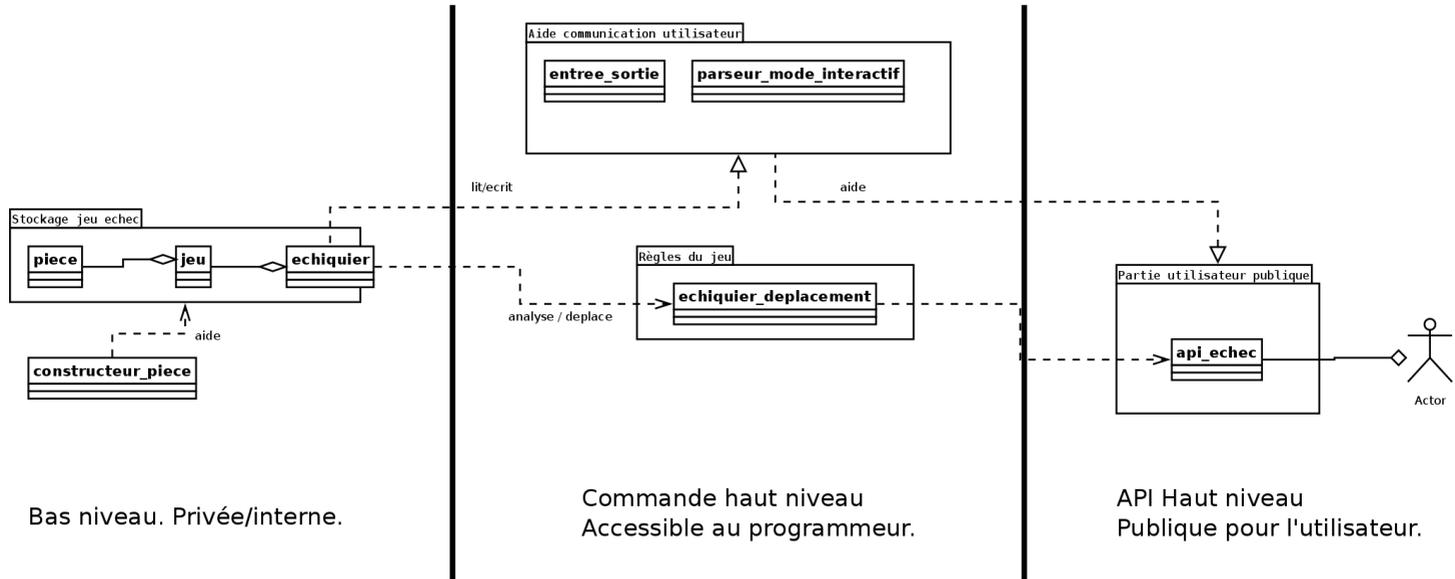


```
typedef struct
{
    jeu jeu_de_piece_blanc;
    jeu jeu_de_piece_noir;
    int joueur_courant;
}echiquier;
```

# Le projet: Déplacement

```
J.0 deplace (6,0) => (5,2)
> d 6 7 -> 5 5
J.1 deplace (6,7) => (5,5)
> d 5 2 -> 5 4
Déplacement impossible
Code erreur 1
> d 5 2 -> 6 4
J.0 deplace (5,2) => (6,4)
```

# Le projet: Organisation globale



# Rappels rapide de C

Fonctions

Passage de paramètres

Organisation de la mémoire

Structs

```
int ma_fonction(int a)
{
    a=1;
}

int main()
{
    int a=2;
    ma_fonction(a);
    printf("%d\n", a);

    return 0;
}
```

# Rappel rapide de C

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

```
int main()
```

```
{
```

```
printf("Hello world\n");
```

```
return 0;
```

```
}
```

une unique fonction  
main par executable



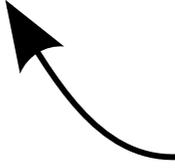
affiche message  
en ligne de commande



# Rappel rapide de C

```
int ma_fonction(int a)
{
    a=1;
}
```

créé une copie de  
l'entier



```
int main()
{
    int a=2;
    ma_fonction(a);
    printf("%d\n", a);

    return 0;
}
```

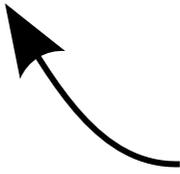
affiche 2



# Rappel rapide de C

```
int ma_fonction(int *a)
{
    *a=1;
}
```

pointeur vers un entier



```
int main()
{
    int a=2;
    ma_fonction(&a);
    printf("%d\n", a);

    return 0;
}
```

adresse de a



affiche 1



# Rappels rapide de C

```
int ma_fonction(int a)
{
    a=1;
}
```

```
int main()
{
```

Fonctions

Passage de paramètres

→ **Organisation de la mémoire**

Structs

```
    int a=2;
    ma_fonction(a);
    printf("%d\n", a);

    return 0;
}
```

# Rappel: Organisation de la mémoire

```
int main()
{
    int variable=0x4147383D;
    char *pointeur_1=&variable;
    float *pointeur_2=&variable;

    printf("0x%x\n",variable);
    printf("%d\n",variable);
    printf("%c %c %c %c\n",*pointeur_1,
        *(pointeur_1+1),
        *(pointeur_1+2),
        *(pointeur_1+3));
    printf("%f\n",*pointeur_2);

    return 0;
}
```

# Rappel: Organisation de la mémoire

```
int main()
{
    int variable=0x4147383D;
    char *pointeur_1=&variable;
    float *pointeur_2=&variable;

    printf("0x%x\n",variable);
    printf("%d\n",variable);
    printf("%c %c %c %c\n",*pointeur_1,
        *(pointeur_1+1),
        *(pointeur_1+2),
        *(pointeur_1+3));
    printf("%f\n",*pointeur_2);

    return 0;
}
```

RAM: (logique)

@100	@101	@102	@103	@104	@105
??	41	47	38	3D	??

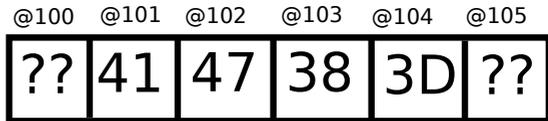
# Rappel: Organisation de la mémoire

```
int main()
{
    int variable=0x4147383D;
    char *pointeur_1=&variable;
    float *pointeur_2=&variable;

    printf("0x%x\n",variable);
    printf("%d\n",variable);
    printf("%c %c %c %c\n",*pointeur_1,
        *(pointeur_1+1),
        *(pointeur_1+2),
        *(pointeur_1+3));
    printf("%f\n",*pointeur_2);

    return 0;
}
```

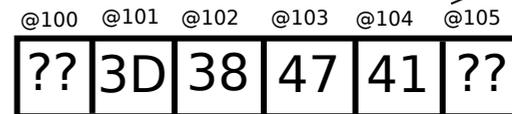
RAM: (logique)



0100 0001 0100 0111 0011 1000 0001 1101

en pratique:  
x86 -> little endian

0001 1101 0011 1000 0100 0111 0100 0001



# Rappel: Organisation de la mémoire

```
int main()
{
    int variable=0x4147383D;
    char *pointeur_1=&variable;
    float *pointeur_2=&variable;

    printf("0x%x\n",variable);
    printf("%d\n",variable);
    printf("%c %c %c %c\n",*pointeur_1,
        *(pointeur_1+1),
        *(pointeur_1+2),
        *(pointeur_1+3));
    printf("%f\n",*pointeur_2);

    return 0;
}
```

RAM:

@100	@101	@102	@103	@104	@105
??	3D	38	47	41	??

```
87654321 0011 2233 4455 6677 8899 aabb ccdd eeff
00000000: 7f45 4c46 0201 0100 0000 0000 0000 0000
00000010: 0100 3e00 0100 0000 0000 0000 0000 0000
00000020: 0000 0000 0000 0000 c801 0000 0000 0000
00000030: 0000 0000 4000 0000 0000 4000 0f00 0c00
00000040: 3078 2578 0a00 2564 0a00 2563 2025 6320
00000050: 2563 2025 630a 0025 660a 0000 0000 0000
00000060: 4883 ec08 ba3d 3847 41be 0000 0000 bf01
00000070: 0000 0031 c0e8 0000 0000 ba3d 3847 41be
00000080: 0000 0000 bf01 0000 0031 c0e8 0000 0000
00000090: 41b9 4100 0000 41b8 4700 0000 b938 0000
000000a0: 00ba 3d00 0000 be00 0000 00bf 0100 0000
000000b0: 31c0 e800 0000 00f2 0f10 0500 0000 00be
000000c0: 0000 0000 bf01 0000 00b8 0100 0000 e800
```

pgm.o

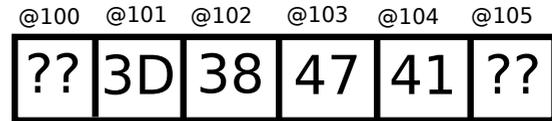
# Rappel: Organisation de la mémoire

```
int main()
{
    int variable=0x4147383D;
    char *pointeur_1=&variable;
    float *pointeur_2=&variable;

    printf("0x%x\n",variable);
    printf("%d\n",variable);
    printf("%c %c %c %c\n",*pointeur_1,
        *(pointeur_1+1),
        *(pointeur_1+2),
        *(pointeur_1+3));
    printf("%f\n",*pointeur_2);

    return 0;
}
```

RAM:



&variable=101

sizeof(int)=4

variable=0x4147383D  
=1095186493

```
.LCFI1:
.cfi_def_cfa_register 6
subq   $32, %rsp
.loc   1 8 0
movl   $1095186493, -20(%rbp)
.loc   1 9 0
```

pgm.s

# Rappel: Organisation de la mémoire

```
int main()
{
    int variable=0x4147383D;
    char *pointeur_1=&variable;
    float *pointeur_2=&variable;

    printf("0x%x\n",variable);
    printf("%d\n",variable);
    printf("%c %c %c %c\n",*pointeur_1,
        *(pointeur_1+1),
        *(pointeur_1+2),
        *(pointeur_1+3));
    printf("%f\n",*pointeur_2);

    return 0;
}
```

RAM:

@100	@101	@102	@103	@104	@105
??	3D	38	47	41	??

pointeur\_1=101  
sizeof(char)=1  
\*pointeur\_1=3D

# Rappel: Organisation de la mémoire

```
int main()
{
    int variable=0x4147383D;
    char *pointeur_1=&variable;
    float *pointeur_2=&variable;

    printf("0x%x\n",variable);
    printf("%d\n",variable);
    printf("%c %c %c %c\n",*pointeur_1,
        *(pointeur_1+1),
        *(pointeur_1+2),
        *(pointeur_1+3));
    printf("%f\n",*pointeur_2);

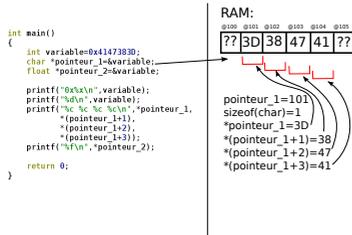
    return 0;
}
```

RAM:



pointeur\_1=101  
sizeof(char)=1  
\*pointeur\_1=3D (=)  
\*(pointeur\_1+1)=38 (8)  
\*(pointeur\_1+2)=47 (G)  
\*(pointeur\_1+3)=41 (A)

# Rappel: Organisation de la mémoire



Note:  
pointeur=>On peut  
parcourir la RAM.

attention sécurité!



```
void hackeur(int *variable);

int main()
{
  //*****//
  //secret!!!
  int code_carte_bleue=0x7941;
  //*****//
```

```
//variable sans rapport
int variable=5;
hackeur(&variable);

return 0;
}
```

```
void hackeur(int *p_variable)
{
  char *p=p_variable;
  printf("code= %x%x\n", *(p+5), *(p+4));
}
```

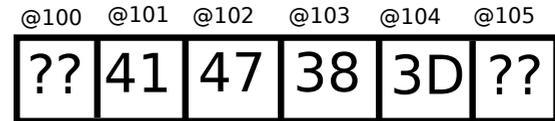
# Rappel: Organisation de la mémoire

```
int main()
{
    int variable=0x4147383D;
    char *pointeur_1=&variable;
    float *pointeur_2=&variable;

    printf("0x%x\n",variable);
    printf("%d\n",variable);
    printf("%c %c %c %c\n",*pointeur_1,
        *(pointeur_1+1),
        *(pointeur_1+2),
        *(pointeur_1+3));
    printf("%f\n",*pointeur_2);

    return 0;
}
```

RAM: (logique)



pointeur\_2=101  
sizeof(float)=4

\*pointeur\_2=

0100 0001 0100 0111 0011 10000 0001 1101  
1 bit | 8 | 17  
bit | exposant | mantisse  
signe

$$N = (-1)^s 2^{(e-127)} (1+f)$$

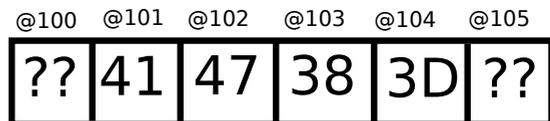
# Rappel: Organisation de la mémoire

```
int main()
{
    int variable=0x4147383D;
    char *pointeur_1=&variable;
    float *pointeur_2=&variable;

    printf("0x%x\n",variable);
    printf("%d\n",variable);
    printf("%c %c %c %c\n",*pointeur_1,
        *(pointeur_1+1),
        *(pointeur_1+2),
        *(pointeur_1+3));
    printf("%f\n",*pointeur_2);

    return 0;
}
```

RAM: (logique)



pointeur\_2=101

sizeof(float)=4

\*pointeur\_2=

0100 0001 0100 0111 0011 10000 0001 1101

1 bit  
signe

8  
exposant

17  
mantisse

$2^{-1}+2^{-5}+2^{-6}+\dots = 0.5563982\dots$

$$N = (-1)^s 2^{(e-127)} (1+f)$$
$$= 1 \quad 2^3 \quad 1.5563982\dots$$
$$= 12.4512$$

# Rappels rapide de C

```
int ma_fonction(int a)
{
    a=1;
}
```

```
int main()
{
```

Fonctions

Passage de paramètres

Organisation de la mémoire

→ **Structs**

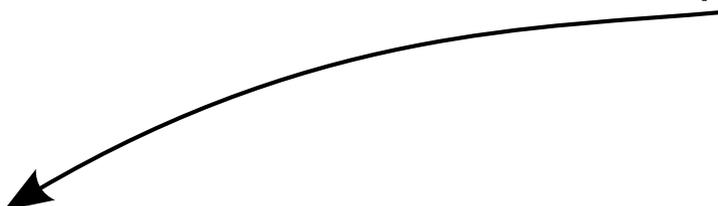
```
int a=2;
ma_fonction(a);
printf("%d\n", a);

return 0;
}
```

# Rappel: Struct

```
struct ma_structure  
{  
    int variable_1;  
    int variable_2;  
    float variable_3;  
    int variable_4;  
};
```

rappel du mot clé struct



```
int main()  
{  
    struct ma_structure a;  
    a.variable_1=5;  
    a.variable_3=12.45;  
  
    struct ma_structure b;  
    b.variable_4=8;  
  
    printf("%f\n", a.variable_3);  
  
    return 0;  
}
```

# Rappel: Struct

```
enum couleur_carrosserie {rouge,vert,bleu,jaune,violet,noir,blanc};
```

```
struct roue_voiture
```

```
{  
    int prix;  
    float pression;  
    char nom_constructeur[16];  
};
```

```
struct carrosserie_voiture
```

```
{  
    int prix;  
    enum couleur_carrosserie couleur;  
};
```

```
struct vitre_voiture
```

```
{  
    int prix;  
};
```

```
struct voiture
```

```
{  
    struct roue_voiture roue[4];  
    struct carrosserie_voiture carrosserie;  
    struct vitre_voiture vitre[6];  
};
```

enumeration



# Rappel: Types de bases (*built-in*)

```
printf("char          (%d)\n", sizeof(char));
printf("short         (%d)\n", sizeof(short));
printf("int           (%d)\n", sizeof(int));
printf("long int      (%d)\n", sizeof(long int));
printf("long long int (%d)\n", sizeof(long long int));
printf("float         (%d)\n", sizeof(float));
printf("double        (%d)\n", sizeof(double));
printf("long double   (%d)\n", sizeof(long double));
printf("void*         (%d)\n", sizeof(void*));
```

<b>char</b>	(1)
short	(2)
<b>int</b>	(4)
long int	(8)
long long int	(8)

entiers signés

unsigned char
unsigned int
unsigned short
unsigned long int
unsigned long long int

entiers positifs

<b>float</b>	(4)
<b>double</b>	(8)
long double	(16)

flottants

char*	(8)
int*	(8)
...	
<b>void*</b>	(8)

pointeurs

# Rappel: Struct multiples

```
enum couleur_carrosserie {rouge,vert,bleu,jaune,violet,noir,blanc};
```

```
struct roue_voiture
```

```
{  
    int prix;  
    float pression;  
    char nom_constructeur[16];  
};
```

```
struct carrosserie_voiture
```

```
{  
    int prix;  
    enum couleur_carrosserie couleur;  
};
```

```
struct vitre_voiture
```

```
{  
    int prix;  
};
```

```
struct voiture
```

```
{  
    struct roue_voiture roue[4];  
    struct carrosserie_voiture carrosserie;  
    struct vitre_voiture vitre[6];  
};
```

```
int main()
```

```
{
```

```
    struct voiture ma_bmw;
```

```
    //prix de la carrosserie  
    ma_bmw.carrosserie=8500;
```

```
    //prix de la roue 2  
    ma_bmw.roue[2].prix=95;
```

```
    //prix de la vitre 0  
    ma_bmw.vitre[0].prix=150;
```

```
    //couleur de la carrosserie  
    ma_bmw.carrosserie.couleur=noir;
```

```
    return 0;
```

```
}
```

# Rappel: Struct (adresse élément)

```
roue_affecte_michelin(struct roue_voiture* roue)
{
    strcpy(roue->nom_constructeur,"Michelin");
    roue->pression=2.1;
    prix=165;
}

int main()
{
    struct voiture ma_bmw;

    //affectation de la roue 2:
    roue_affecte_michelin( &(amp;ma_bmw.roue[2]) );

    return 0;
}
```



↑  
adresse de la roue

# Rappel: Struct anonyme

```
typedef struct
{
    float prix;
    float quantite;
}stock_pomme;
```

permet d'éviter la répétition du mot struct



```
int main()
{
    stock_pomme mon_stock;
    mon_stock.prix=1.14;
    mon_stock.quantite=3.14;

    return 0;
}
```

# Gestion code important

- IDE
- Séparation en-tête/implémentation
- Gestionnaire de version

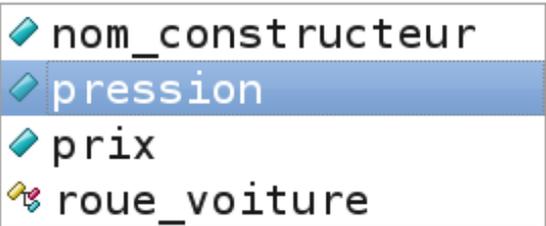
# Utilisation d'un IDE

*Integrated  
Development  
Environments*

Evite perte de temps  
Complétion automatique  
Rappel mémoire

```
int main()
{
    struct voiture ma_bmw;

    ma_bmw.roue[2].
    return 0;
}
```



*IDE Linux:*  
**QtCreator**  
**Eclipse**  
**Code::Blocks**

*Editeur texte:*  
Vi, Emacs, gedit, ...



# Gestion code important

IDE

→ **Séparation en-tête/implémentation**

Gestionnaire de version

# Fichiers en-tête

Signature/en tête d'une fonction

Implémentation / corps d'une fonction

# Fichiers en-tête

## Solution 1:

```
int somme(int tableau[], unsigned int taille)
{
    int somme_courante=0;

    unsigned int k=0;
    for(k=0;k<taille;++k)
        somme_courante += tableau[k];

    return somme_courante;
}

int main()
{
    int T[]={1,4,5,7};

    int s=somme(T, sizeof(T)/sizeof(int));
    printf("%d\n", s);

    return 0;
}
```

## Solution 2:

```
int somme(int tableau[], unsigned int taille);

int main()
{
    int T[]={1,4,5,7};

    int s=somme(T, sizeof(T)/sizeof(int));
    printf("%d\n", s);

    return 0;
}

int somme(int tableau[], unsigned int taille)
{
    int somme_courante=0;

    unsigned int k=0;
    for(k=0;k<taille;++k)
        somme_courante += tableau[k];

    return somme_courante;
}
```

# Fichiers en-tête

## Solution 3:



fichier.h

```
int somme(int tableau[], unsigned int taille)
```



fichier.c

```
int somme(int tableau[], unsigned int taille)
{
    int somme_courante=0;

    unsigned int k=0;
    for(k=0;k<taille;++k)
        somme_courante += tableau[k];

    return somme_courante;
}
```



main.c

```
#include <fichier.h>
int main()
{
    int T[]={1,4,5,7};

    int s=somme(T, sizeof(T)/sizeof(int));
    printf("%d\n", s);

    return 0;
}
```

# Fichiers en-tête

## Synthèse:

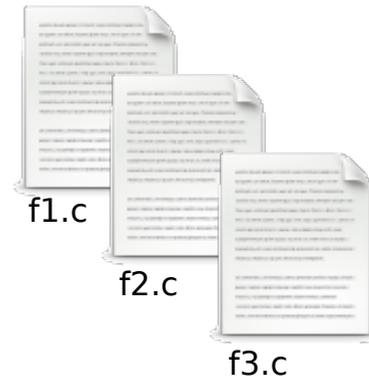
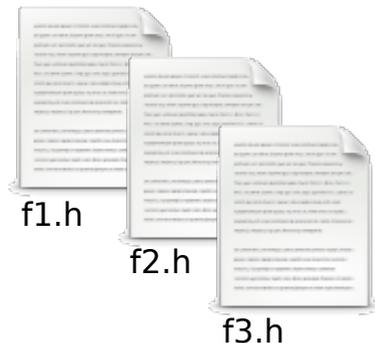
- + Evite les longs fichiers
- + Séparation en-tete / implémentation

publique  
entree/sortie  
stable

privée  
peut changer

## En pratique:

Autant de fichiers que d'abstractions

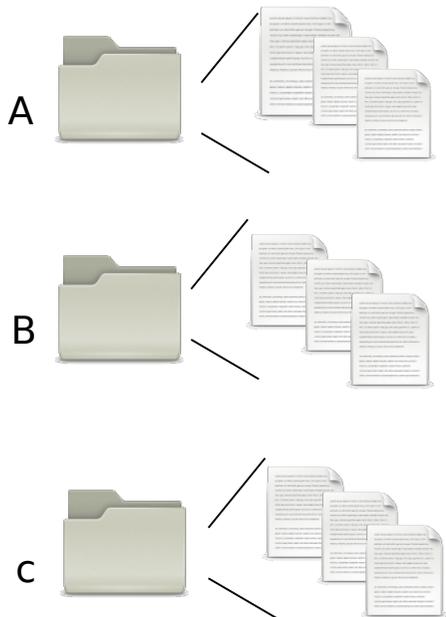


# Fichiers en-tête

Synthèse:

Pour les très gros projets:

ex. LibC



# Gestion code important

IDE

Séparation en-tête/implémentation

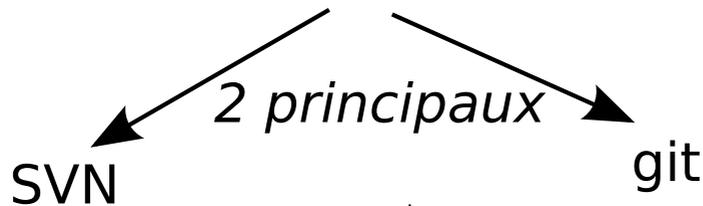
→ **Gestionnaire de version**

# Programmer à plusieurs

~~Parties spécifiques~~ *inter-dépendance*

Passage par mails *petit code uniquement*

Logiciel de controle de version (pro)

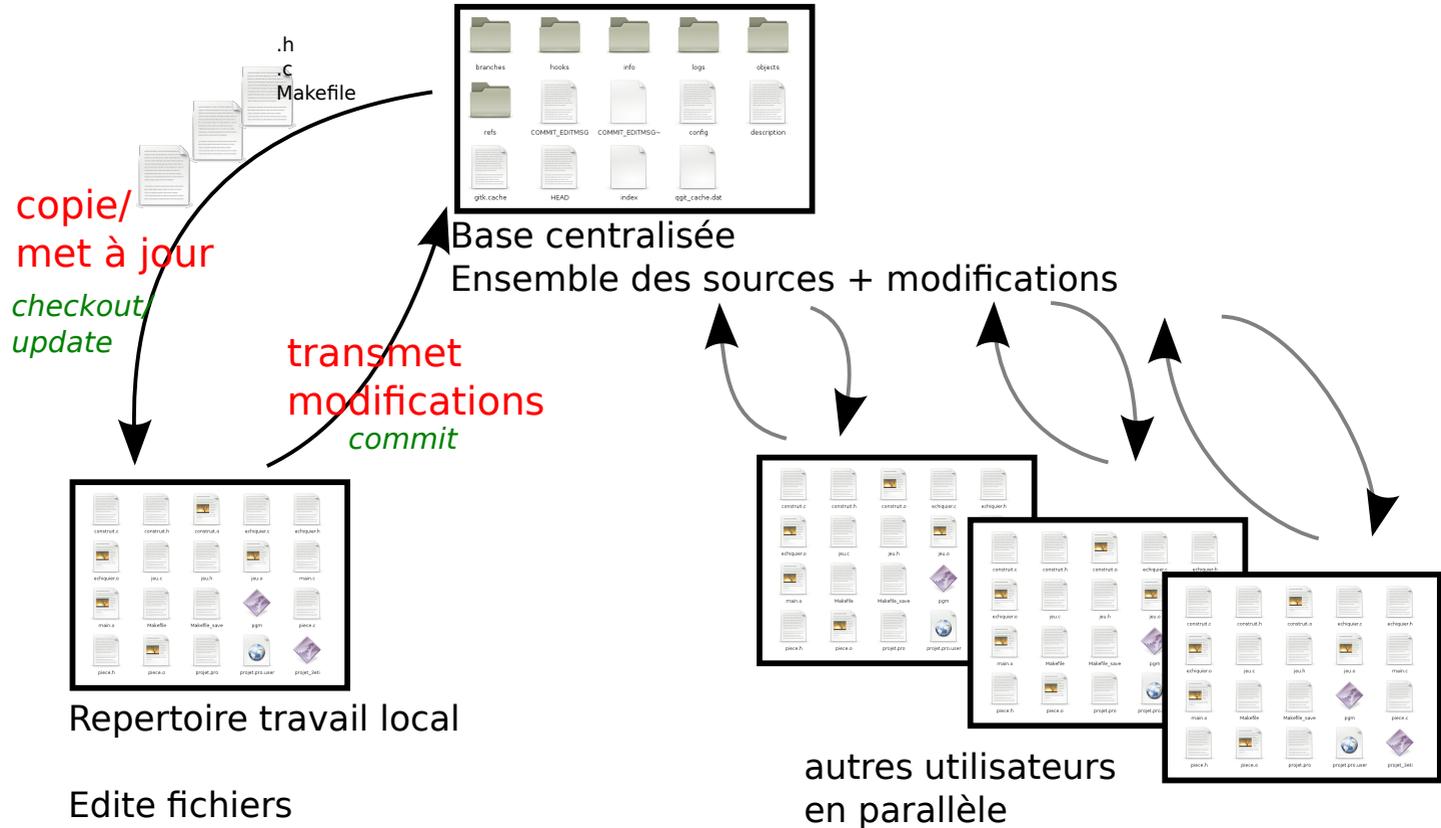


plus ancien  
uniquement centralisé  
- nécessite un serveur

récent  
centralisé ou distribué

# Programmer à plusieurs

## Principe d'un logiciel de controle de version



# Programmer à plusieurs

## Principe d'un logiciel de controle de version

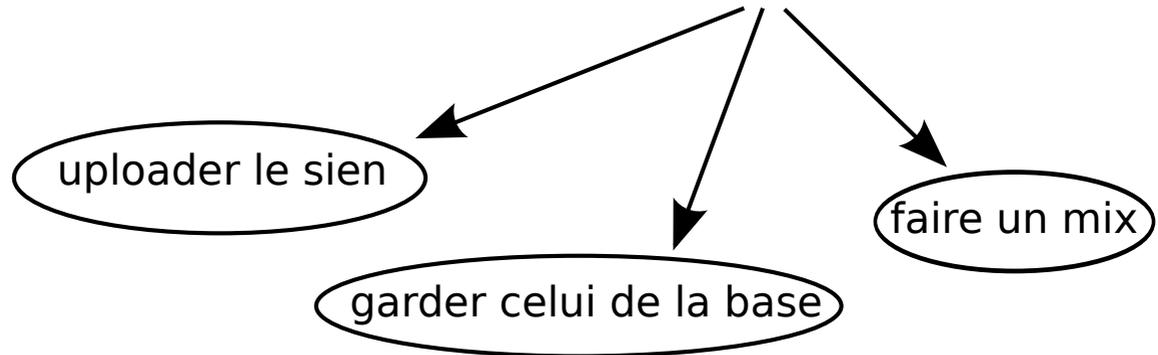
A chaque *commit*:

Si nouvelle version mise à jour entre temps

*entre update et commit*

Modifications indépendantes => pas de conflits

Modifications d'un même contenu => choix

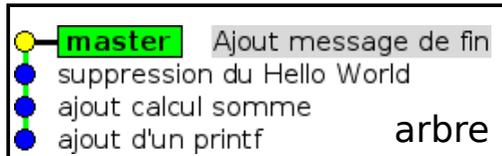


# Programmer à plusieurs

## Principe d'un logiciel de controle de version

La base garde en mémoire l'ensemble des modifications!

- + On ne perd aucune version
- + On peut revenir en arrière



Fichier local:

```
#include <stdio.h>

int calcul_somme(int a,int b)
{
    return a+b;
}

int main()
{
    int a=3;
    int b=8;
    printf("%d\n",calcul_somme(a,b));
    printf("Fin\n");
    return 0;
}
```

## Modifications enregistrées

```
@@ -0,0 +1,7 @@
+#include <stdio.h>
+
+int main()
+{
+    printf("Hello world\n");
+    return 0;
+}
```

1

```
@@ -1,7 +1,17 @@
#include <stdio.h>

+int calcul_somme(int a,int b)
+{
+    return a+b;
+}
+
int main()
{
    printf("Hello world\n");
+
+    int a=3;
+    int b=8;
+    printf("%d\n",calcul_somme(a,b));
+
    return 0;
}
```

2

```
@@ -7,8 +7,6 @@ int calcul_somme(int a,int b)
int main()
{
-    printf("Hello world\n");
-
    int a=3;
    int b=8;
    printf("%d\n",calcul_somme(a,b));
}
```

3

```
@@ -11,5 +11,7 @@ int main()
    int b=8;
    printf("%d\n",calcul_somme(a,b));
+
+    printf("Fin\n");
+
    return 0;
}
```

4

# Programmer à plusieurs

Principe d'un logiciel de controle de version.

Mode d'emploi:

```
$ emacs mon_fichier.c #creation d'un/plusieurs fichiers sources
$ git init             #initialisation du repertoire .git
$ git add mon_fichier.c #ajout du suivit du fichier désigné
$ emacs mon_fichier.c #Modification du/des fichiers sources ...
$ git commit -a        #upload des modifications
```

<http://git-scm.com/book/en/Git-Basics-Getting-a-Git-Repository>

**github**  
SOCIAL CODING



# Licences logiciels

# Logiciel Libre

Un logiciel libre est-il gratuit?

Un freeware est-il un logiciel libre?

Un shareware est-il un logiciel libre?

Un logiciel open-source est-il un logiciel libre?

Peut-on revendre un logiciel libre?

Puis-je intégrer du code libre dans le mien?

Peut-on réutiliser à son nom du code d'un logiciel libre?

# Logiciel Libre

Logiciel libre = définition de Richard Stallman  
(*mouvement de pensée*)



*créateur de la FSF*



The Free Software Foundation (FSF) is a nonprofit with a worldwide mission to promote computer user freedom and to defend the rights of all free software users.

L'expression « logiciel libre » veut dire que le logiciel respecte la liberté de l'utilisateur et de la communauté.  
En gros, les utilisateurs ont la liberté d'exécuter, de copier, de distribuer, d'étudier, de modifier et d'améliorer le logiciel.  
Avec ces libertés, les utilisateurs (à la fois individuellement et collectivement) contrôlent le programme et ce qu'il fait pour eux.

Un programme est un logiciel libre si vous, en tant qu'utilisateur de ce programme, avez les quatre libertés essentielles :

- 0/ La liberté d'exécuter le programme, pour tous les usages (liberté 0) ;
- 1/ La liberté d'étudier le fonctionnement du programme, et de le modifier pour qu'il effectue vos tâches informatiques comme vous le souhaitez (liberté 1) ;  
l'accès au code source est une condition nécessaire ;
- 2/ La liberté de redistribuer des copies, donc d'aider votre voisin (liberté 2) ;
- 3/ La liberté de distribuer aux autres des copies de vos versions modifiées (liberté 3) ;  
en faisant cela, vous donnez à toute la communauté une possibilité de profiter de vos changements ;  
l'accès au code source est une condition nécessaire.

**Note:** Il existe des musiques libres, films libres, art libre, ...

# Logiciel Open Source

Logiciel dont la licence respecte les critères de l'Open Source Initiative



**définition:** <http://opensource.org/docs/osd>

Pas uniquement code source disponible (mais ambiguïté existe)

**Note:** *Logiciel Libre => OpenSource*  
*Mais on peut être OpenSource et non Libre*

## Licences Open Source

<http://opensource.org/licenses/category>

- :: License that are popular and widely used or with strong communities ::
  - Apache License 2.0 (Apache-2.0)
  - BSD 3 Clause "New" or "Revised" license (BSD-3-Clause)
  - BSD 4 Clause "Simplified" or "FreeBSD" license (BSD-2-Clause)
  - GNU General Public License (GPL)
  - GNU Library or "Lesser" General Public License (LGPL)
  - MIT license (MIT)
  - Mozilla Public License 2.0 (MPL-2.0)
  - Common Development and Distribution License (CDL-1.0)
  - Eclipse Public License (EPL-1.0)
- :: Special purpose licenses ::
  - Educational Community License
  - IPA Font License (IPA)
  - NASA Open Source Agreement 1.3 (NASA-1.3)
  - Open Font License 1.1 (OFL-1.1)
- :: Other/Miscellaneous licenses ::
  - Adaptive Public License (APL-1.0)
  - Artistic License 2.0 (Artistic-2.0)
  - Open Software License (OSL-3.0)
  - Q Public License (QPL-1.0)
  - zlib/libpng license (Zlib)
- :: Licenses that are redundant with more popular licenses ::
  - Academic Free License (AFL-3.0)
  - Attribution Assurance Licenses (AAL)
  - CERN Forum License (2.0) (CFL-2.0)
  - Fair License (Fair)
  - Historical Permission Notice and Disclaimer (HPND)
  - Lucent Public License Version 1.02 (LPL-1.02)
  - The PostgreSQL License (PostgreSQL)
  - University of Illinois/NCSA Open Source License (IUCS/A)
  - X-11 License (X11)
- :: Non-reusable licenses ::
  - Apple Public Source License (APSL-2.0)
  - Computer Associates Trusted Open Source License 1.1 (CATOSL-1.1)

# Licences Open Source

Il existe plusieurs licences open sources.

*Licence*=condition d'utilisation et diffusion du logiciel et de son code.

Licences issues de la FSF

## Les plus répandues:

<http://www.gnu.org/copyleft/gpl.html>

### GPL

Utilisation code GPL => totalité du logiciel licence GPL  
Ne peut pas être utilisé dans un projet sous copyright

```
<one line to give the program's name and a brief idea of what it does.>  
Copyright (c) <year> <name of author>
```

```
This program is free software: you can redistribute it and/or modify  
it under the terms of the GNU General Public License as published by  
the Free Software Foundation, either version 3 of the License, or  
(at your option) any later version.
```

```
This program is distributed in the hope that it will be useful,  
but WITHOUT ANY WARRANTY; without even the implied warranty of  
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the  
GNU General Public License for more details.
```

```
You should have received a copy of the GNU General Public License  
along with this program. If not, see <http://www.gnu.org/licenses/>.
```

### LGPL

Utilisation code LGPL => n'implique pas que l'ensemble soit LGPL  
Peut être utilisé pour une projet sous copyright

### BSD

### MIT

**Note** pour vidéos, arts, ...: licences Creative Common

# Licences Copyright vs Copyleft

Licences fermées copyright: (all right reserved)

Vous n'avez pas le droit de redistribuer sous aucune forme.  
Demande explicite nécessaire.

Attention: recopie de passage de site internet = illégale

ex. **Copyright © 1997-2011 Cprogramming.com. All rights reserved.**

Attention:

Freeware, Shareware = licence sous copyright

# Libre et gratuit

Libre et/ou OpenSource ne signifie pas gratuit!

Nombreux projets commerciaux:

Red Hat

GNAT (compilateur ADA)

MySQL Enterprise

NetBeans

Zimbra

...

Distribution gratuite ou payante  
Service généralement payant  
Entreprise fournissant un service

*Oracle*  
*Mandriva*  
*Mozilla*

...

# Droits d'auteurs

## Droit morale

Paternité  
Choix de diffusion  
...

Inaliénable, Imprescriptible

=> Il est interdit (et impossible)  
de changer le nom de  
l'auteur original!!

## Droit patrimoniale

Privilège d'exploitation  
(royalties, ...)

Note:  
En France pas de brevets logiciels!

# Qualité du code

*Qu'est ce qu'un bon code?*

# Qualité du code

Qu'est ce qu'un bon code?

un code en un minimum de lignes ?

un code avec un maximum d'optimisations ?

un code qui n'utilise que des pointeurs ?

un code qui se lit simplement ?

un code qui s'écrit simplement ?

un code qui montre sa complexité ?

un code qui cache sa complexité ?

# Bonnes pratiques

Qu'est ce qu'un bon code?

Code A:

```
int main()
{
    int v1[]={1,2,3,-1},v2[]={4,5,6,-1};
    int *p1=v1,*p2=v2;
    while(*(p1++)!=-1) *(p1-1)+=*(p2++);
```

Code B:

```
int calcul_longueur(int vecteur[])
{
    int taille_entier=sizeof(int);
    int longueur=sizeof(vecteur)/taille_entier;
    return longueur;
}

int main()
{
    int v1[]={1,2,3};
    int v2[]={4,5,6};

    int longueur_vecteur=calcul_longueur(v1);

    int resultat[longueur_vecteur];
    int k=0;
    for(k=0;k<longueur_vecteur;k++)
    {
        resultat[k] = v1[k] + v2[k];
    }
}
```

# Bonnes pratiques

Quel est le code le plus rapide? le plus lisible?

```
void init(int v[],int N)
{
    unsigned int k=0;
    for(k=0;k<N;++k)
        v[k]=k;
}

void fonction(int v1[],int v2[],int v3[],int N);

int main()
{
    int N=16;
    int v1[N]; int v2[N]; int v3[N];

    init(v1,N); init(v2,N);  init(v3,N);

    fonction(v1,v2,v3,N);
    return 0;
}
```

# Bonnes pratiques

Quel est le code le plus rapide? le plus lisible?

```
void init(int v1[],int N)
{
    unsigned int k=0;
    for(k=0;k<N;++k)
        v1[k]=k;
}

void fonction(int v1[],int v2[],int v3[],int N);

int main()
{
    int N=16;
    int v1[N]; int v2[N]; int v3[N];
    init(v1,N); init(v2,N); init(v3,N);
    fonction(v1,v2,v3,N);
    return 0;
}
```

1

```
void fonction(int v1[],int v2[],int v3[],int N)
{
    unsigned int k=0;
    for(k=1;k<N-1;++k)
    {
        v1[k]+=3;
        v3[k]+=v1[k-1]-v2[k+1];
        v3[k]/=10;
    }
}
```

2

```
void fonction(int v1[],int v2[],int v3[],int N)
{
    int *p_v1=v1,*p_v1_2=v1+1;
    int *p_v2=v2+2;
    int *p_v3=v3+1;

    register unsigned int k=1;
    while(k<N-1)
    {
        *(p_v1_2++) += 3;
        *p_v3 += *(p_v1++)-*(p_v2++);
        (*p_v3)/=10; ++p_v3;
        ++k;
    }
}
```

3

```
void fonction(int v1[],int v2[],int v3[],int N)
{
    v1[1]+=3;
    v3[1]+=(v1[0]-v2[2]);v3[1]/=10;

    v1[2]+=3;
    v3[2]+=(v1[1]-v2[3]);v3[2]/=10;

    v1[3]+=3;
    v3[3]+=(v1[2]-v2[4]);v3[3]/=10;

    v1[4]+=3;
    v3[4]+=(v1[3]-v2[5]);v3[4]/=10;

    v1[5]+=3;
    v3[5]+=(v1[4]-v2[6]);v3[5]/=10;

    v1[6]+=3;
    v3[6]+=(v1[5]-v2[7]);v3[6]/=10;

    v1[7]+=3;
    v3[7]+=(v1[6]-v2[8]);v3[7]/=10;

    v1[8]+=3;
    v3[8]+=(v1[7]-v2[9]);v3[8]/=10;

    v1[9]+=3;
    v3[9]+=(v1[8]-v2[10]);v3[9]/=10;

    v1[10]+=3;
    v3[10]+=(v1[9]-v2[11]);v3[10]/=10;
}
```

4

```
void fonction(int v1[],int v2[],int v3[],int N)
{
    int v1_00=v1[0],v2_00=v2[0];
    int v1_01=v1[1],v2_01=v2[1];
    int v1_02=v1[2],v2_02=v2[2];
    int v1_03=v1[3],v2_03=v2[3];
    int v1_04=v1[4],v2_04=v2[4];
    int v1_05=v1[5],v2_05=v2[5];
    int v1_06=v1[6],v2_06=v2[6];
    int v1_07=v1[7],v2_07=v2[7];
    int v1_08=v1[8],v2_08=v2[8];
    int v1_09=v1[9],v2_09=v2[9];
    int v1_10=v1[10],v2_10=v2[10];
    int v1_11=v1[11],v2_11=v2[11];
    int v1_12=v1[12],v2_12=v2[12];
    int v1_13=v1[13],v2_13=v2[13];
    int v1_14=v1[14],v2_14=v2[14];
    int v1_15=v1[15],v2_15=v2[15];

    v1[1]+=3;
    v3[1]+=(v1_00-v2_02);v3[1]/=10;

    v1[2]+=3;
    v3[2]+=(v1_01-v2_03);v3[2]/=10;

    v1[3]+=3;
    v3[3]+=(v1_02-v2_04);v3[3]/=10;

    v1[4]+=3;
    v3[4]+=(v1_03-v2_05);v3[4]/=10;

    v1[5]+=3;
    v3[5]+=(v1_04-v2_06);v3[5]/=10;

    v1[6]+3;
}
```

# Bonnes pratiques

## Quel est le code le plus rapide? le plus lisible?

```
void init(int v1[],int N)
{
    unsigned int k=0;
    for(k=0;k<N;++k)
        v1[k]=k;
}

void fonction(int v1[],int v2[],int v3[],int N);

int main()
{
    int N=16;
    int v1[N]; int v2[N]; int v3[N];
    init(v1,N); init(v2,N); init(v3,N);
    return fonction(v1,v2,v3,N);
}
```

Pour 75 000 000 executions:

1 245ms

```
void fonction(int v1[],int v2[],int v3[],int N)
{
    unsigned int k=0;
    for(k=1;k<N-1;++k)
    {
        v1[k]+=3;
        v3[k]+=v1[k-1]-v2[k+1];
        v3[k]/=10;
    }
}
```

2 245ms

```
void fonction(int v1[],int v2[],int v3[],int N)
{
    int *p_v1=v1,*p_v1_2=v1+1;
    int *p_v2=v2+2;
    int *p_v3=v3+1;

    register unsigned int k=1;
    while(k<N-1)
    {
        *(p_v1_2++) += 3;
        *p_v3 += *(p_v1++)-*(p_v2++);
        (*p_v3)/=10; ++p_v3;
        ++k;
    }
}
```

3 245ms

```
void fonction(int v1[],int v2[],int v3[],int N)
{
    v1[1]+=3;
    v3[1]+=(v1[0]-v2[2]);v3[1]/=10;

    v1[2]+=3;
    v3[2]+=(v1[1]-v2[3]);v3[2]/=10;

    v1[3]+=3;
    v3[3]+=(v1[2]-v2[4]);v3[3]/=10;

    v1[4]+=3;
    v3[4]+=(v1[3]-v2[5]);v3[4]/=10;

    v1[5]+=3;
    v3[5]+=(v1[4]-v2[6]);v3[5]/=10;

    v1[6]+=3;
    v3[6]+=(v1[5]-v2[7]);v3[6]/=10;

    v1[7]+=3;
    v3[7]+=(v1[6]-v2[8]);v3[7]/=10;

    v1[8]+=3;
    v3[8]+=(v1[7]-v2[9]);v3[8]/=10;

    v1[9]+=3;
    v3[9]+=(v1[8]-v2[10]);v3[9]/=10;

    v1[10]+=3;
    v3[10]+=(v1[9]-v2[11]);v3[10]/=10;
}
```

4 245ms

```
void fonction(int v1[],int v2[],int v3[],int N)
{
    int v1_00=v1[0],v2_00=v2[0];
    int v1_01=v1[1],v2_01=v2[1];
    int v1_02=v1[2],v2_02=v2[2];
    int v1_03=v1[3],v2_03=v2[3];
    int v1_04=v1[4],v2_04=v2[4];
    int v1_05=v1[5],v2_05=v2[5];
    int v1_06=v1[6],v2_06=v2[6];
    int v1_07=v1[7],v2_07=v2[7];
    int v1_08=v1[8],v2_08=v2[8];
    int v1_09=v1[9],v2_09=v2[9];
    int v1_10=v1[10],v2_10=v2[10];
    int v1_11=v1[11],v2_11=v2[11];
    int v1_12=v1[12],v2_12=v2[12];
    int v1_13=v1[13],v2_13=v2[13];
    int v1_14=v1[14],v2_14=v2[14];
    int v1_15=v1[15],v2_15=v2[15];

    v1[1]+=3;
    v3[1]+=(v1_00-v2_02);v3[1]/=10;

    v1[2]+=3;
    v3[2]+=(v1_01-v2_03);v3[2]/=10;

    v1[3]+=3;
    v3[3]+=(v1_02-v2_04);v3[3]/=10;

    v1[4]+=3;
    v3[4]+=(v1_03-v2_05);v3[4]/=10;

    v1[5]+=3;
    v3[5]+=(v1_04-v2_06);v3[5]/=10;

    v1[6]+3;
}
```

# Bonnes pratiques

Quel est le code le plus rapide? le plus lisible?

Pour 75 000 000 executions :



```
void init(int v[],int N)
{
    unsigned int k=0;
    for(k=0;k<N;++k)
        v[k]=k;
}

void fonction(int v1[],int v2[],int v3[],int N)
{
    int main()
    {
        int N=16;
        int v1[N]; int v2[N]; int v3[N];
        init(v1,N); init(v2,N); init(v3,N);
        fonction(v1,v2,v3,N);
        return 0;
    }
}
```

Si on réfléchit:  
v3 indépendant de v1 et v2.  
Il existe une solution analytique

```
int c=75000000;
v3[0]=0;
v3[N-1]=N-1;
unsigned int k=0;
for(k=1;k<N-1;++k)
    v3[k]=(c-1)/3;
```

temps: 0ms

# Bonnes pratiques

Remarque optimisation:

*(années 80)*

On optimise pas le code !

*inliner*  
*déboucler*  
*register*  
=> travail du **compilateur**

On optimise l'algorithme

# Bonnes pratiques

Remarque optimisation 2:

GCC optimise très bien

```
int main()
{
    int a=0;
    int k=0;

    for(k=0;k<112;++k)
        a += 4*k;

    printf("%d\n",a);
}
```

```
main:
.LFB0:
    .cfi_startproc
    movl    $24864, %esi
    movl    $.LC0, %edi
    xorl    %eax, %eax
    jmp     printf
    .cfi_endproc
```

code assembleur  
après \$ gcc -O2

plus de boucle  
 $O(1)$

# Bonnes pratiques

Remarque optimisation 2:

GCC optimise très bien  
souvent mieux qu'un humain!

ex. somme coefficient d'une matrice:

```
int main()
{
    int matrice[3][3]={{1,2,3},
                       {4,1,-5},
                       {7,7,1}};

    int somme=0;

    int kx=0,ky=0;
    for(kx=0;kx<3;++kx)
        for(ky=0;ky<3;++ky)
            somme += matrice[kx][ky];

    printf("%d\n",somme);
}
```

```
main:
.LFB0:
    .cfi_startproc
    movl    $21, %esi
    movl    $.LC0, %edi
    xorl    %eax, %eax
    jmp     printf
    .cfi_endproc
```

déjà calculé  
O(1)

\$ gcc -O2

# Bonnes pratiques

Remarque optimisation 2:

GCC optimise très bien  
souvent mieux qu'un humain!

ex. somme coefficient d'une matrice:

```
int main()
{
    int matrice[3][3]={{1,2,3},
                       {4,1,-5},
                       {7,7,1}};

    int somme=0;

    int *p=matrice;
    register int c=0;
    while(c++ < 9)
        somme += *(p++);

    printf("%d\n",somme);
}
```

- lisible  
- rapide !!

```
main:
.LFB0:
.cfi_startproc
movl    $1, -56(%rsp)
movl    $2, -52(%rsp)
movl    $.LC0, %edi
movl    $3, -48(%rsp)
movl    $4, -44(%rsp)
xorl    %eax, %eax
movdqa  -56(%rsp), %xmm0
movl    $1, -40(%rsp)
movl    $-5, -36(%rsp)
movl    $7, -32(%rsp)
movl    $7, -28(%rsp)
padd    -40(%rsp), %xmm0
movdqa  %xmm0, %xmm1
movl    $1, -24(%rsp)
psrldq  $8, %xmm1
padd    %xmm1, %xmm0
movdqa  %xmm0, %xmm1
psrldq  $4, %xmm1
padd    %xmm1, %xmm0
movd    %xmm0, -60(%rsp)
movl    -60(%rsp), %esi
addl    $1, %esi
jmp     printf
.cfi_endproc
```

\$ gcc -O2

# Bonnes pratiques

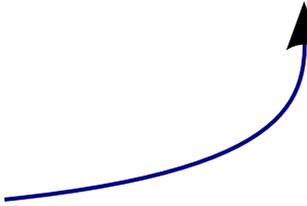
Synthèse:

**Bon code** = code qui se **lit simplement**



Suit les règles standards  
Cache sa complexité  
Documenté et s'auto-documente

Contraintes  
supplémentaires



généricité  
efficacité  
portabilité  
légèreté  
fiabilité

# Méthodologies de conception

Développement par contrat

Développement par algorithmes

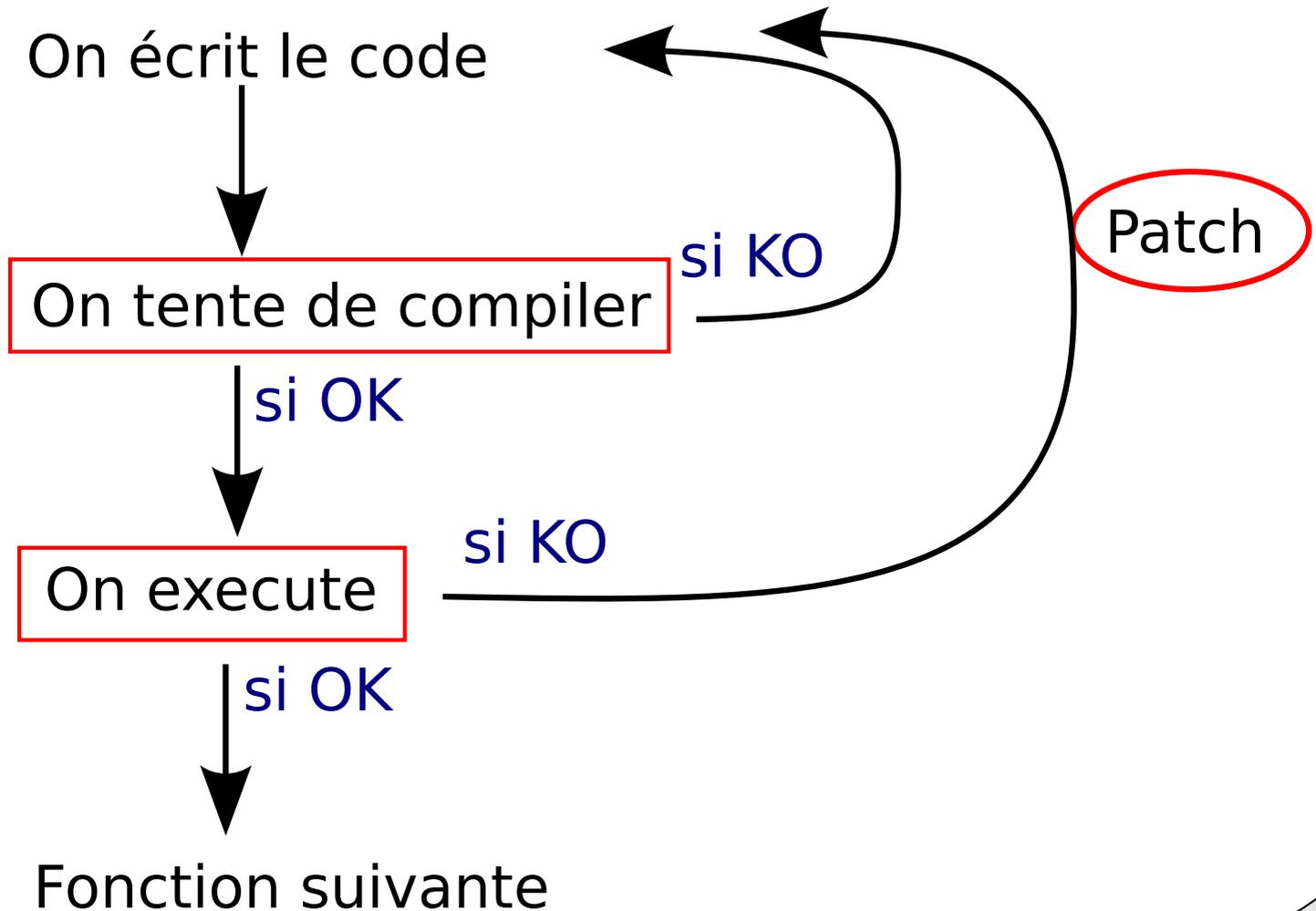
Développement par tests

# Comment écrire un bon code ?

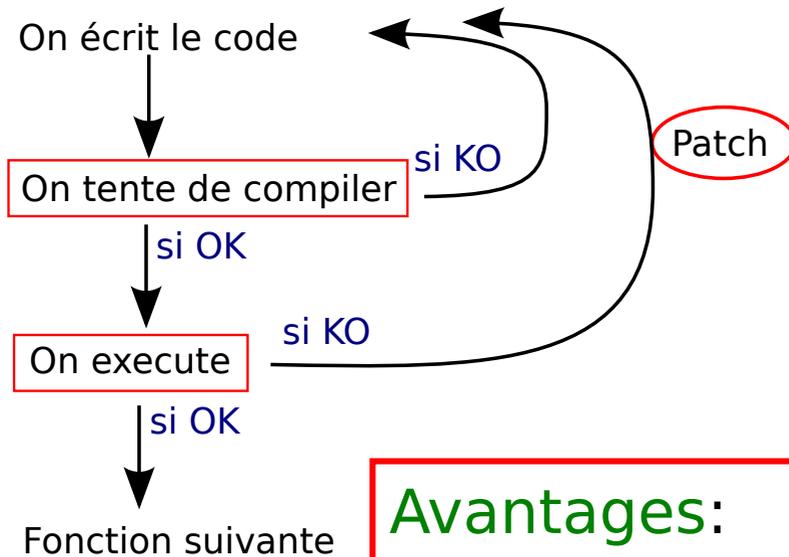
Plusieurs méthodologies de conception:

- Développement par **contrats**
- Développement par **tests**
- Développement par **algorithmes**
- Développement par hack

# Developpement par hack



# Developpement par hack



## Avantages:

Minimise effort intellectuel à court terme

## Inconvénients:

Patch dès le début

=> Difficile à lire (machine qui guide)

=> Ingérable pour des gros projets

# Méthodologies de conception

## → **Développement par contrat**

Développement par algorithmes

Développement par tests

# Developpement par contrats

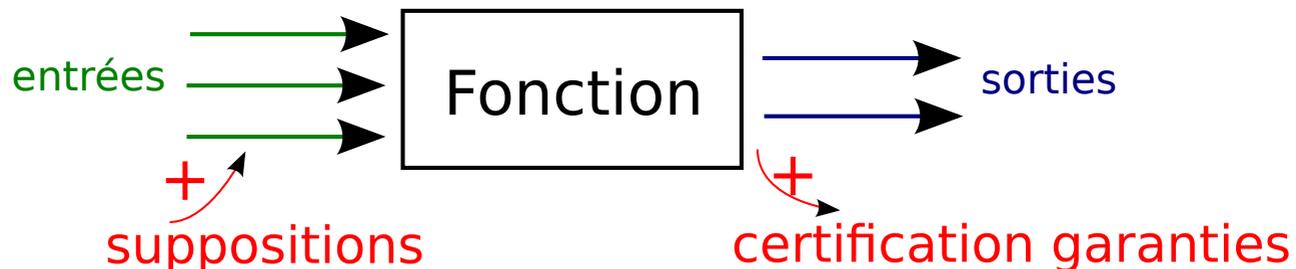
**Avant** d'écrire l'implémentation d'une fonction

On définit:

contrat avec le programmeur

1- Les **suppositions** sur les arguments d'entrées  
(ou état d'un système)

2- Les **certifications** après execution de la fonction



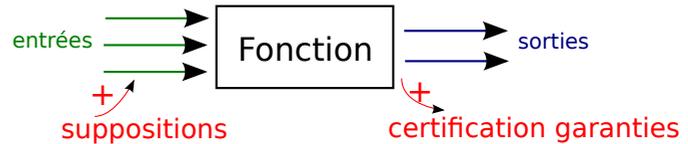
# Developpement par contrats

**Avant** d'écrire l'implémentation d'une fonction

On définit:



- 1- Les **suppositions** sur les arguments d'entrées (ou état d'un système)
- 2- Les **certifications** après execution de la fonction



## Avantages:

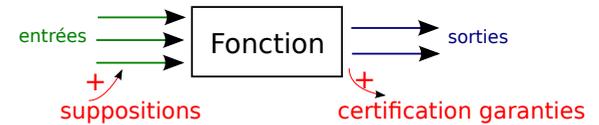
- On empêche les cas particuliers/effets de bords  
=> on ne les oublie pas
- Auto-documente la fonction
- Aide aux tests, garantit la validité  
=> réduction des bugs

## Inconvénients:

- Travail supplémentaire en amont

# Developpement par contrats

Exemple:



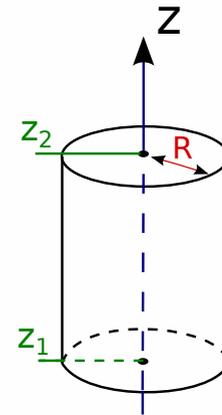
```
#include <stdio.h>
#include <math.h>

//Calcul le volume du cylindre de rayon R entre z=[z1,z2]
float volume(float z1,float z2,float R)
{
    return 2*M_PI*R*(z2-z1);
}

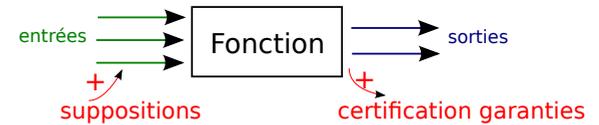
int main()
{
    float V1=volume(1,2,0.5);
    float V2=volume(0,10,8);
    float V3=volume(-4,8,1);
    float V4=volume(5,1,1);
    float V5=volume(1,4,-1);

    return 0;
}
```

Fonction OK?



# Developpement par contrats



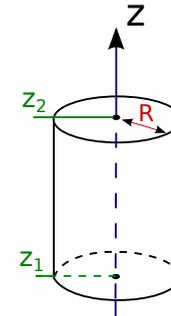
Exemple:

```
/*Calcul le volume du cylindre de rayon R entre z=[z1,z2]
 * Le volume est obtenu par la formule  $V=2 \pi (z2-z1) R$ .
 *
 * Necessite:
 * - deux coordonnées flottantes (z1,z2) avec  $z1 < z2$ 
 * - une coordonnée flottante  $R > 0$ 
 * Garantie:
 * - renvoi un volume (flottant positif) correspondant au cylindre decrit.
 */
float volume(float z1,float z2,float R)
{
  assert(z2>z1);
  assert(R>0);

  float V=2*M_PI*R*(z2-z1);

  assert(V>0);
  return V;
}
```

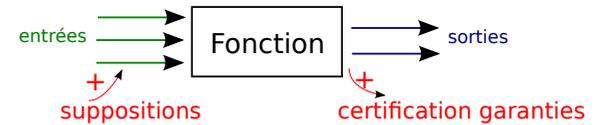
contrat  
avec le programmeur



si non vérifiée:  
quitte en indiquant la ligne

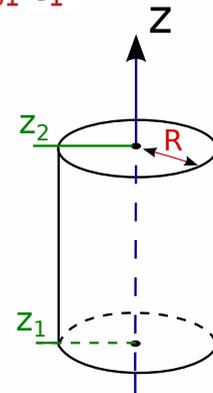
# Developpement par contrats

Exemple:



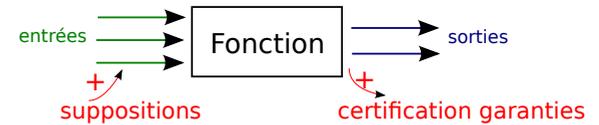
```
/*Calcul le volume du cylindre de rayon R entre z=[z1,z2]
 * Le volume est obtenu par la formule V=2 pi (z2-z1) R.
 *
 * Necessite:
 * - deux coordonnées flottantes (z1,z2) quelconques
 * - une coordonnée flottante R quelconque
 * Garantie:
 * - Si z1>z2 et R>0, renvoi un volume (flottant positif) correspondant au cylindre decrit.
 * - Si z1=z2 ou R=0, renvoi zero
 * - Si z1<z2 ou R<0, affiche une erreur en ligne de commande et renvoi -1
 */
float volume(float z1,float z2,float R)
{
  //cas du volume nul
  float epsilon=1e-5;
  if(fabs(z1-z2)<epsilon || fabs(R)<epsilon)
    return 0.0;
  //cas non geometrique
  if(z1<z2 || R<0)
    {printf("Erreur calcul volume\n");return -1.0;}
  //volume du cylindre
  float V=2*M_PI*R*(z2-z1);
  return V;
}
```

← autre contrat possible avec le programmeur



# Developpement par contrats

Synthèse:



+ Auto-documentation:

(le *code* est  
la documentation)

```
/*Calcul le volume du cylindre de rayon R entre z=[z1,z2]
 * Le volume est obtenu par la formule V=2 pi (z2-z1) R.
 *
 * Necessite:
 * - deux coordonnées flottantes (z1,z2) avec z1<z2
 * - une coordonnée flottante R>0
 * Garantie:
 * - renvoi un volume (flottant positif) correspondant au cylindre decrit.
 */
```

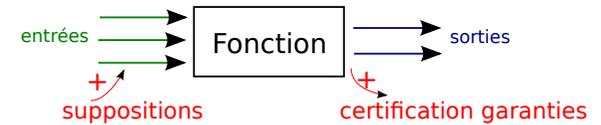
+ Cas particuliers gérés:  
(pas d'oublis)

```
float V5=volume(1,4,-1);
```

- de debug
  - explications, +lisible
- => gain de temps au final

# Developpement par contrats

## Exemple 2:



```
/* Calcul de la moyenne d'un ensemble de valeurs comprises entre 0 et 100
 * La moyenne est approxime sur l'entier le plus proche
 *
 * Necessite:
 * - Un pointeur constant vers un ensemble de valeurs entieres. Pointeur non NULL.
 *   les valeurs du tableau sont comprises entre 0 et 100
 * - Un entier positif indiquant la taille du tableau de donnees.
 * Garantie:
 * - Renvoi la moyenne des valeurs du tableau (compris entre 0 et 100).
 * - La moyenne est approximee au nombre entier le plus proche.
 */
int moyenne(const int* valeurs,unsigned int taille);
```

contrat

à mettre dans l'en-tête

```
int moyenne(const int* valeurs,unsigned int taille)
{
    assert(valeurs!=NULL);

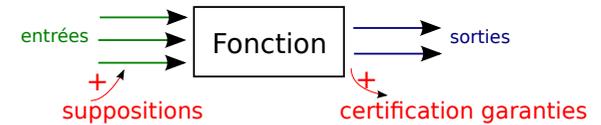
    //somme de l'ensemble des valeurs
    int moyenne=0;
    int k=0;
    for(k=0;k<taille;++k)
    {
        int val=valeurs[k];
        assert(val>=0 && val<=100);
        moyenne += valeurs[k];
    }

    //passage en nombre flottants temporaires (virgules)
    float temporaire=(float)moyenne/taille;
    //approximation sur l'entier le plus proche
    moyenne=(int)(temporaire+0.5);

    return moyenne;
}
```

implémentation

# Developpement par contrats



Proposition de syntaxe:

```
/**
 * Fonction NOM_FONCTION
 * *****
 *   DECRIT BUT DE LA FONCTION
 *
 *   Necessite:
 *   - DECRIT LES CONTRATS SUR LES VARIABLES ET ETAT DU SYSTEME
 *   Garantie:
 *   - DECRIT LES GARANTIES LORSQUE LES CONTRATS SONT RESPECTEES
 */
type_retour nom_fonction(type_var1 variable1, type_var2 variable2, ...);
```

A mettre dans le fichier d'en-tête : documentation

# Programmation défensive?

Passer un contrat avec le programmeur?  
Mais programmeur = vous?

Vous êtes la 1ère source d'erreur / bugs  
Protégez vous contre vous même!

```
assert()  
if() { }  
...
```

# Méthodologies de conception

Développement par contrat

→ **Développement par algorithmes**

Développement par tests

# Programmation par algorithme

**Avant** d'écrire l'implémentation d'une fonction

On écrit:

**L'algorithme** dans un langage humain

Attention: algorithme  $\neq$  code

# Programmation par algorithme

Exemple:

```
/* Calcul de la moyenne d'un ensemble de valeurs comprises entre 0 et 100
 * La moyenne est approxime sur l'entier le plus proche
 */
int moyenne(const int* valeurs,unsigned int taille)
{
    //Algorithme: calcul de la moyenne
    //
    // Variable moyenne <- 0
    //
    // Pour toutes les valeurs v du tableau
    //   verifier v compris entre [0,100]
    //   ajouter v a moyenne
    // Fin Pour
    //
    // Diviser moyenne par la taille du tableau
    // Arrondir a l'entier le plus proche
    //
    // Renvoyer valeur de la moyenne
    //
}
```

# Programmation par algorithme

Exemple:

```
/* Calcul de la moyenne d'un ensemble de valeurs comprises entre 0 et 100
 * La moyenne est approxime sur l'entier le plus proche
 */
int moyenne(const int* valeurs,unsigned int taille)
{
    //Algorithme: calcul de la moyenne
    //
    // Variable moyenne <- 0
    //
    // Pour toutes les valeurs v du tableau
    // verifier v compris entre [0,100]
    // ajouter v a moyenne
    // Fin Pour
    //
    // Diviser moyenne par la taille du tableau
    // Arrondir a l'entier le plus proche
    //
    // Renvoyer valeur de la moyenne
    //
}
```

complétion

```
/* Calcul de la moyenne d'un ensemble de valeurs comprises entre 0 et 100
 * La moyenne est approxime sur l'entier le plus proche
 */
int moyenne(const int* valeurs,unsigned int taille)
{
    assert(valeurs!=NULL);

    // Variable moyenne <- 0
    int moyenne=0;

    // Pour toutes les valeurs v du tableau
    unsigned int k=0;
    for(k=0;k<taille;++k)
    {
        // verifier v compris entre [0,100]
        int v=valeurs[k];
        assert(v<=0 && v>=100);
        // ajouter v a moyenne
        moyenne += v;
    }

    // Diviser moyenne par la taille du tableau
    float temporaire=(float)moyenne/taille;
    // Arrondir a l'entier le plus proche
    moyenne=(int)(temporaire+0.5);

    // Renvoyer valeur de la moyenne
    return moyenne;
}
```

# Programmation par algorithme

## Exemple:

```
void recherche_minimum(float x0, float y0)
{
    float x=x0;
    float y=y0;
    float e=1e-5;

    float dx[8]={e,e,0,-e,-e,-e,0,e};
    float dy[8]={0,e,e,e,0,-e,-e,-e};

    float v=fonction_inconnue(x,y);
    float min=v;
    int k_opt=-1;
    float n[8];

    unsigned int k=0;
    do
    {
        v=fonction_inconnue(x,y);
        min=v;
        k_opt=-1;
        for(k=0;k<8;++k)
        {
            n[k]=fonction_inconnue(x+dx[k],y+dy[k]);
            if(min>n[k])
            {
                min=n[k];
                k_opt=k;
            }
        }
        if(k_opt!=-1)
        {
            x+=dx[k_opt];
            y+=dy[k_opt];
        }
    }
    while(k_opt!=-1);

    printf("f(%f %f)=%f\n",x,y,v);
}
```

??

```
int main()
{
    recherche_minimum(1,2);
}

float fonction_inconnue(float x, float y)
{
    return exp(-cos(x-y)+x*x+y*y)+(0.5-sqrt(x*y))*exp(-x*x-0.5*y*y);
}
```

# Programmation par algorithme

Exemple:

```
void recherche_minimum(float x0, float y0)
{
    //*****
    //Recherche de minimum de fonction_inconnue
    //*****
    //
    // Algorithme:
    //
    // Initialise (x,y) a (x0,y0)
    //
    // Pre-stocker la table de deplacement pour 8 voisins
    // C'est a dire: deplacement[k]={ (-dx,+dy)[3] ( 0,+dy)[2] (+dx,+dy)[1] }
    //                               { (-dx, 0)[4] (+dx, 0)[0] }
    //                               { (-dx,-dy)[5] ( 0,-dy)[6] (+dx,-dy)[7] }
    //
    // Tant que minimum non atteint
    //
    // Affecter a V0 la valeur de fonction_inconnue(x,y)
    // Pour tous les 8 voisins [k] de (x,y)
    //
    // Soit (x2,y2) le voisin courant de (x,y)
    // C'est a dire: x2=x +/- dx
    //              y2=y +/- dy
    //
    // Affecter a V[k] la valeur de fonction_inconnue(voisin courant)
    //
    // Fin Pour
    //
    // Cherchez le minimum de V[k] pour k=k_optimal
    //
    // Si V[k_optimal] plus petit que V0
    // Alors avancer (x,y) dans la direction du voisin k_optimal
    // Fin Si
    //
    //
    // Fin Tant que
}
```

# Programmation par algorithme

## Exemple:

```
void recherche_minimum(float x0,float y0)
{
    //*****
    //Recherche de minimum de fonction_inconnue
    //*****
    //
    // Algorithme:
    //
    // Initialise (x,y) a (x0,y0)
    float x=x0;
    float y=y0;

    // Prestocker la table de deplacement pour 8 voisins
    // C'est a dire: deplacement[k]=(-dx,+dy)[3] ( 0,+dy)[2] (+dx,+dy)[1] }
    // { (-dx, 0)[4] (+dx, 0)[0] }
    // { (-dx,-dy)[5] ( 0,-dy)[6] (+dx,-dy)[7] }
    float deplacement[8][2]={dx,0},{dx,dy},{0,dy},{-dx,+dy},{-dx,0},{-dx,-dy},{0,-dy},{+dx,-dy}};

    //
    // Tant que minimum non atteint
    //
    int minimum_atteint=0;
    while(minimum_atteint==0)
    {
        // Affecter a V0 la valeur de fonction_inconnue(x,y)
        float V0=fonction_inconnue(x,y);

        // Pour tous les 8 voisins [k] de (x,y)
        float V[8];
        unsigned int k=0;
        for(k=0;k<8;++k)
        {
            //
            // Soit (x2,y2) le voisin courant de (x,y)
            // C'est a dire: x2=x +/- dx
            // y2=y +/- dy
            //
            float x2=x+deplacement[k][0];
            float y2=y+deplacement[k][1];
```

```
        // Affecter a V[k] la valeur de fonction_inconnue(voisin courant)
        //
        V[k]=fonction_inconnue(x2,y2);
        // Fin Pour
        //
        // Cherchez le minimum de V[k] pour k=k_optimal
        int k_optimal=trouve_min(V,8);

        //
        // Si V[k_optimal] plus petit que V0
        // Alors avancer (x,y) dans la direction du voisin k_optimal
        // Sinon
        // minimum_atteint est vrai
        // Fin Si

        if(V[k_optimal]<V0)
        {
            x += deplacement[k_optimal][0];
            y += deplacement[k_optimal][1];
        }
        else
        {
            minimum_atteint=1;
        }
    }
}
```

```
//trouve le minimum d'un vecteur
int trouve_min(float V[],int taille)
{
    assert(taille>=1);
    // Initialiser minimum au premier element du vecteur
    // Initialiser indice k_optimal a 0
    //
    // Pour tout les elements d'indice [k] restants du vecteur V
    // Si minimum > V[k]
    // Alors affecter minimum=V[k],
    // et sauvegarder k_optimal=k
    // Fin Si
    // Fin Pour
    // Retourner k_optimal

    float minimum=V[0];
    int k_optimal=0;

    unsigned int k=0;
    for(k=1;k<taille;++k)
    {
        if(minimum>V[k])
        {
            minimum=V[k];
            k_optimal=k;
        }
    }

    return k_optimal;
}
```

# Programmation par algorithme

## Synthèse

+ Documentation du code automatique

+ Aide à l'écriture du code

+ lisible

+ code meilleur qualité

# Méthodologies de conception

Développement par contrat

Développement par algorithmes

→ **Développement par tests**

# Programmation par algorithme

Ne pas confondre algorithme et code

ex. **A NE PAS FAIRE!**

```
int moyenne(const int* valeurs,unsigned int taille)
{
    //Algorithme: calcul de la moyenne
    //
    // Variable moyenne=0
    // Variable k=0
    // Pour k=0, tant que k<taille, k++
    //  assert valeurs[k]<0 && valeurs[k]<100
    //  moyenne += valeurs[k]
    // Fin Pour
    //
    // moyenne /= taille;
    // moyenne=(int)(moyenne+0.5)
    //
    // return moyenne
}
```

```
int moyenne(const int* valeurs,unsigned int taille)
{
    //Algorithme: calcul de la moyenne
    //
    // Variable moyenne=0
    int moyenne=0;
    // Variable k=0
    int k=0;
    // Pour k=0, tant que k<taille, k++
    for(k=0;k<taille;++k)
    {
        //  assert valeurs[k]<0 && valeurs[k]<100
        assert(valeurs[k]>=0 && valeurs[k]<=100);
        //  moyenne += valeurs[k]
        moyenne+=valeurs[k];
        // Fin Pour
    }
    //
    // moyenne /= taille;
    float temporaire=(float)moyenne/taille;
    // moyenne=(int)(moyenne+0.5)
    moyenne=(int)(temporaire+0.5);
    //
    // return moyenne
    return moyenne;
}
```

réécriture mots pour mots  
les commentaires n'apportent aucune amélioration de  
lisibilité au code

# Programmation par tests

**Avant** d'écrire l'implémentation d'une fonction

On écrit:

Les tests que doivent satisfaire la fonction

*En général:*

On connaît ce que doit faire la fonction  
avant de connaître son code



# Programmation par tests

## Exemple:

```
//1 valeur, attend 1
int t1[]={50};
if(moyenne(t1,sizeof(t1)/sizeof(int))!=50)
    ECHEC_TEST;

//arrondie au nombre superieur, attend 72
int t2[]={45,80,90};
if(moyenne(t2,sizeof(t2)/sizeof(int))!=72)
    ECHEC_TEST;

//arrondi au nombre inferieur, attend 71
int t3[]={45,80,89};
if(moyenne(t3,sizeof(t3)/sizeof(int))!=71)
    ECHEC_TEST;

//valeurs constantes, attend 45
int t4[]={45,45,45,45,45};
if(moyenne(t4,sizeof(t4)/sizeof(int))!=45)
    ECHEC_TEST;
```

tests différents cas:  
valides + invalides

fixe entrée  
=> attend sortie spécifique

```
//valeurs constantes, attend 45
int t4[]={45,45,45,45,45};
if(moyenne(t4,sizeof(t4)/sizeof(int))!=45)
    ECHEC_TEST;

//cas particulier 0 valeurs, attend 0
int t5[0];
if(moyenne(t5,sizeof(t5)/sizeof(int))!=0)
    ECHEC_TEST;

//cas particulier depassement valeur < 0, attend -1
int t6[]={5,-4};
if(moyenne(t6,sizeof(t6)/sizeof(int))!=-1)
    ECHEC_TEST;

//cas particulier depassement valeur > 100, attend -1
int t7[]={8,106};
if(moyenne(t7,sizeof(t7)/sizeof(int))!=-1)
    ECHEC_TEST;

//cas particulier limite depassement valeur < 0, attend -1
int t8[]={1,2,-1};
if(moyenne(t8,sizeof(t8)/sizeof(int))!=-1)
    ECHEC_TEST;

//cas limite, valeur==0, attend 10
int t9[]={10,20,0};
if(moyenne(t9,sizeof(t9)/sizeof(int))!=10)
    ECHEC_TEST;

//cas limite particulier, valeur==101, attend -1
int t10[]={8,101,99};
if(moyenne(t10,sizeof(t10)/sizeof(int))!=-1)
    ECHEC_TEST;

//cas limite, valeur==100, attend 69
int t11[]={8,100,99};
if(moyenne(t11,sizeof(t11)/sizeof(int))!=69)
    ECHEC_TEST;
```

# Programmation par tests

Exemple:

```
//1 valeur, attend 1
int t1[]={50};
if(moyenne(t1,sizeof(t1)/sizeof(int))!=50)
    ECHEC_TEST;

//arrondie au nombre superieur, attend 72
int t2[]={45,80,90};
if(moyenne(t2,sizeof(t2)/sizeof(int))!=72)
    ECHEC_TEST;

//arrondi au nombre inferieur, attend 71
int t3[]={45,80,89};
if(moyenne(t3,sizeof(t3)/sizeof(int))!=71)
    ECHEC_TEST;

//valeurs constantes, attend 45
int t4[]={45,45,45,45,45};
if(moyenne(t4,sizeof(t4)/sizeof(int))!=45)
    ECHEC_TEST;

//valeurs constantes, attend 45
int t4[]={45,45,45,45,45};
if(moyenne(t4,sizeof(t4)/sizeof(int))!=45)
    ECHEC_TEST;

//cas particulier 0 valeurs, attend 0
int t5[0];
if(moyenne(t5,sizeof(t5)/sizeof(int))!=0)
    ECHEC_TEST;

//cas particulier depassement valeur < 0, attend -1
int t6[]={5,-4};
if(moyenne(t6,sizeof(t6)/sizeof(int))!=-1)
    ECHEC_TEST;

//cas particulier depassement valeur > 100, attend -1
int t7[]={8,106};
if(moyenne(t7,sizeof(t7)/sizeof(int))!=-1)
    ECHEC_TEST;

//cas particulier limite depassement valeur < 0, attend -1
int t8[]={1,2,-1};
if(moyenne(t8,sizeof(t8)/sizeof(int))!=-1)
    ECHEC_TEST;

//cas limite, valeur=0, attend 10
int t9[]={10,20,0};
if(moyenne(t9,sizeof(t9)/sizeof(int))!=10)
    ECHEC_TEST;

//cas limite particulier, valeur=101, attend -1
int t10[]={8,101,99};
if(moyenne(t10,sizeof(t10)/sizeof(int))!=-1)
    ECHEC_TEST;

//cas limite, valeur=100, attend 69
int t11[]={8,100,99};
if(moyenne(t11,sizeof(t11)/sizeof(int))!=69)
    ECHEC_TEST;
```

```
int moyenne(const int* valeurs,unsigned int taille)
{
    assert(valeurs!=NULL);

    //cas particulier taille==0
    if(taille==0)
        return 0;

    int moyenne=0;
    unsigned int k=0;
    for(k=0;k<taille;++k)
    {
        int v=valeurs[k];
        //cas d'erreur
        if(v<0 || v>100)
            return -1;

        moyenne += v;
    }

    float temporaire=(float)moyenne/taille;
    // Arrondir a l'entier le plus proche
    moyenne=(int)(temporaire+0.5);

    return moyenne;
}
```

valide

implémentation possible

# Programmation par tests

Exemple:

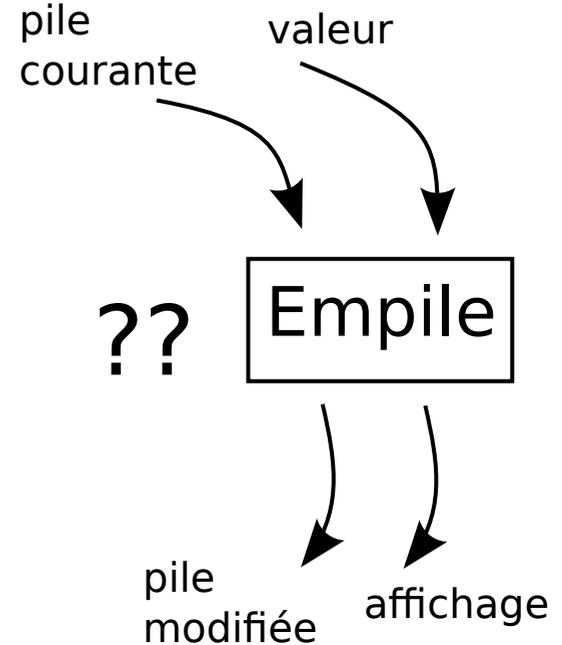
```
#define TAILLE_MAX 15

struct pile
{
    int indice;
    int buffer[TAILLE_MAX];
};

void initialise(struct pile* p)
{
    p->indice=0;
}

int depile(struct pile* p)
{
    if(p->indice>0)
        return p->buffer[--p->indice];
    printf("Erreur liste vide\n");
    return 0;
}

int est_vide(const struct pile* p)
{
    return p->indice==0;
}
```



# Programmation par tests

## Exemple:

```
#define TAILLE_MAX 15
struct pile
{
    int indice;
    int buffer[TAILLE_MAX];
};
void initialise(struct pile* p)
{
    p->indice=0;
}
int depile(struct pile* p)
{
    if(p->indice>0)
        return p->buffer[--p->indice];
    printf("Erreur liste vide\n");
    return 0;
}
int est_vide(const struct pile* p)
{
    return p->indice==0;
}
```

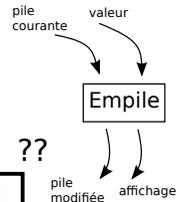
```
/**/
//Test Pile
/**/
struct pile p;
initialise(&p);

empile(&p,5);
int var_0=depile(&p); //doit retourner 5
if(var_0!=5){printf("Erreur %d\n",__LINE__);}

empile(&p,6);
empile(&p,7);
int var_1=depile(&p); //doit retourner 7
int var_2=depile(&p); //doit retourner 6
if(var_1!=7 || var_2!=6){printf("Erreur %d\n",__LINE__);}

int var_3=est_vide(&p); //doit retourner vrai
if(var_3!=1) {printf("Erreur %d\n",__LINE__);}

int k=0;
for(k=0;k<TAILLE_MAX;k++)
    empile(&p,k);
empile(&p,5); //doit afficher une erreur: vecteur plein
int var_4=depile(&p); //doit retourner TAILLE_MAX-1
if(var_4!=TAILLE_MAX-1) {printf("Erreur %d\n",__LINE__);}
```



# Programmation par tests

## Exemple:

```
#define TAILLE_MAX 15
struct pile
{
  int indice;
  int buffer[TAILLE_MAX];
};
void initialise(struct pile* p)
{
  p->indice=0;
}
int depile(struct pile* p)
{
  if(p->indice>0)
    return p->buffer[--p->indice];
  printf("Erreur liste vide\n");
  return 0;
}
int est_vide(const struct pile* p)
{
  return p->indice==0;
}
```

```
//*****//
//Test Pile
//*****//
struct pile p;
initialise(&p);

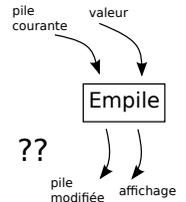
empile(&p,5);
int var_0=depile(&p); //doit retourner 5
if(var_0!=5){printf("Erreur %d\n",__LINE__);}

empile(&p,6);
empile(&p,7);
int var_1=depile(&p); //doit retourner 7
int var_2=depile(&p); //doit retourner 6
if(var_1!=7 || var_2!=6){printf("Erreur %d\n",__LINE__);}

int var_3=est_vide(&p); //doit retourner vrai
if(var_3!=1) {printf("Erreur %d\n",__LINE__);}

int k=0;
for(k=0;k<TAILLE_MAX;k++)
  empile(&p,k);
empile(&p,5); //doit afficher une erreur: vecteur plein
int var_4=depile(&p); //doit retourner TAILLE_MAX-1
if(var_4!=TAILLE_MAX-1) {printf("Erreur %d\n",__LINE__);}
```

```
void empile(struct pile* p,int a)
{
  if(p->indice<TAILLE_MAX)
    p->buffer[p->indice++]=a;
  else
    printf("Erreur buffer plein\n");
}
```



pile courante valeur

pile modifiée

affichage

# Programmation par tests

## Synthèse:

+ Aide à définir l'interface de la fonction (paramètres, entrées/sorties)

*(pas d'ajout de paramètres une fois le corps écrit)*

+ Gère les cas particuliers

*(pas d'oublis)*

+ Debug de la fonction immédiate: plus besoin de réécrire des tests

*=> Assure la validité de la fonction*

+ Gagne du temps sur l'écriture des tests  
(qui doivent avoir lieu plus tard sinon)

+ Valider les fonctionnalités à tout moment.

*Code qui ne régresse pas*

# Organisation des données

## *Pointeurs/Structs*

Declaration  
Passage de paramètres  
Mot clé **const**

# Déclaration structs

explicite

```
struct v3
{
    float x;
    float y;
    float z;
};

int main()
{
    struct v3 mon_vecteur_3d;
}
```

struct anonyme

```
typedef struct
{
    float x;
    float y;
}v2;

int main()
{
    v2 mon_vecteur_2d;
}
```

# Bonne pratique: documentation

Documentez un maximum vos déclarations de structs

```
//Structure contenant un vecteur de coordonnees 3D
struct v3
{
    float x; //coordonnee x
    float y; //coordonnee y
    float z; //coordonnee z
};
```

*roles,  
plage de variations,  
valeurs typiques,  
explications, ...*



```
#define TAILLE_MAX 50 //taille maximal d'un nom

//Structure contenant les informations d'un etudiant
struct etudiant
{
    char nom[TAILLE_MAX]; //Nom de l'etudiant
    char prenom[TAILLE_MAX]; //Prenom de l'etudiant

    int classe;//0 correspond a 3ETI
                //1 correspond a 4ETI
                //2 correspond a 5ETI

    float moyenne;//sa moyenne
                //nombre flottant a 3 decimales
                //compris entre 0.0 et 20.0
};
```

# Importance struct+fonctions

## Importance des structs + fonctions

```
//Structure contenant un vecteur de coordonnees 3D
struct v3
{
    float x; //coordonnee x
    float y; //coordonnee y
    float z; //coordonnee z
};

//Affecte les coordonnees (x,y,z) au vecteur
void v3_set(struct v3* vec, float x, float y, float z);
//Renvoie la norme du vecteur
float v3_norm(const struct v3* vec);
//Affiche les coordonnees du vecteur en ligne de commande
void v3_print(const struct v3* vec);

int main()
{
    struct v3 mon_vecteur_3d;
    v3_set(&mon_vecteur_3d,1,2,2);
    float n=v3_norm(&mon_vecteur_3d);

    printf("||");
    v3_print(&mon_vecteur_3d);
    printf("||");
    printf("=%f\n",n);
}
```

fonctions  
agissants sur une struct



## Implémentation

```
void v3_set(struct v3* vec, float x, float y, float z)
{
    vec->x=x;
    vec->y=y;
    vec->z=z;
}
float v3_norm(const struct v3* vec)
{
    return sqrt(vec->x*vec->x+vec->y*vec->y+vec->z*vec->z);
}
void v3_print(const struct v3* vec)
{
    printf("(%f,%f,%f)", vec->x, vec->y, vec->z);
}
```

# Importance struct+fonctions

Modélise une **abstraction**

*vecteur  
voiture / roues, ...*

- + Cache la complexité
- + Manipulation aisée des données

# Organisation des données

## *Pointeurs/Structs*

Declaration

→ **Passage de paramètres**

Mot clé **const**

# Passage de paramètres

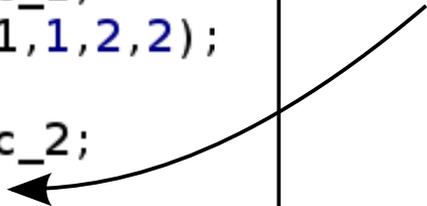
## Passage de paramètres

```
int main()
{
    struct v3 vec_1;
    v3_set(&vec_1,1,2,2);

    struct v3 vec_2;
    vec_2=vec_1;
    vec_2.y=3;

    v3_print(&vec_1);
    v3_print(&vec_2);
}
```

copie



copie: struct a=b

Pour tout les champs <k>  
a.<k>=b.<k>  
Fin Pour

# Passage de paramètres

```
void fonction(struct v3 a)
{
    a.x=8;
}

int main()
{
    struct v3 vec_1={1,1,5};
    fonction(vec_1);
    printf("%d",vec_1.x);
}
```

Passage par copie

# Passage de paramètres

```
float v8_fonction(struct v8 a)
{
    return a.x0+a.x1;
}

int main()
{
    struct v8 vec_1={1,1,5,1,4,5,7,8};
    for(k=0;k<5000000;++k)
    {
        v8_fonction(vec_1);
    }
}
```

Copie réellement  
nécessaire ?

# Passage de paramètres

```
float v8_fonction(struct v8* a)
{
    return a->x0+a->x1;
}

int main()
{
    struct v8 vec_1={1,1,5,1,4,5,7,8};
    for(k=0;k<5000000;++k)
    {
        v8_fonction(&vec_1);
    }
}
```

Passage par pointeur  
= 8 octets < sizeof(v8)

↑  
32 octets

# Passage de paramètres

## Passage de paramètres



```
float v3_norm(struct v3* a);  
  
int main()  
{  
    struct v3 vec_1={1,1,5};  
    float n=v3_norm(&vec_1);  
    printf("%f\n",n);  
}
```

Pointeur

=>danger d'intégrité

Comment détecter/éviter?



```
float v8_norm(struct v3* a)  
{  
    float n=sqrt(a->x*a->x+a->y*a->y+a->z*a->z);  
    a->x=152;  
    return n;  
}
```

# Passage de paramètres

Mot clé **const**

```
float v3_norm(const struct v3* a);
```

Assure que a n'est pas modifié  
Pas besoin de regarder l'implémentation

# Passage de paramètres

Mot clé **const**

```
float v3_norm(const struct v3* a);
```

Assure que a n'est pas modifié  
Pas besoin de regarder l'implémentation

implémentation identique

```
float v3_norm(const struct v3* a)
{
    float n=sqrt(a->x*a->x+a->y*a->y+a->z*a->z);
    return n;
}
```

# Organisation des données

## *Pointeurs/Structs*

Declaration

Passage de paramètres

→ **Mot clé `const`**

# Mot clé **const**

En cas d'erreur:

```
float v3_norm(const struct v3* a)
{
    float n=sqrt(a->x*a->x+a->y*a->y+a->z*a->z);
    a->x=153;
    return n;
}
```

compilation



```
gcc struct.c -lm -g -lrt
struct.c: In function 'v3_norm':
struct.c:52:5: error: assignment of member 'x' in read-only object
```

Evite de mal coder!!

# Mot clé **const**: utilisation

*const se propage*

```
struct CD_musique
{
    char titre[50];
    char auteur[50];
    int prix;
};
struct DVD_film
{
    char titre[50];
    int duree;
    int prix;
};
struct etagere
{
    struct CD_musique CD[100];
    struct DVD_film DVD[100];
};
```

# Mot clé **const**: utilisation

const se *propage*

```
struct CD_musique
{
    char titre[50];
    char auteur[50];
    int prix;
};
struct DVD_film
{
    char titre[50];
    int duree;
    int prix;
};
struct etagere
{
    struct CD_musique CD[100];
    struct DVD_film DVD[100];
};
```

```
//renvoie pointeur vers CD sur mon etagere
CD_musique* retourne_CD_musique(etagere* mon_etagere,int indice)
{
    return &(amp; mon_etagere->CD[indice] );
}
//renvoie pointeur (constant) vers CD sur mon etagere
const CD_musique* cherche_CD_musique(const etagere* mon_etagere,int indice)
{
    return &(amp; mon_etagere->CD[indice] );
}
```

# Mot clé **const**: utilisation

const se *propage*

```
//renvoie pointeur vers CD sur mon etagere
CD_musique* retourne_CD_musique(etagere* mon_etagere, int indice)
{
    return &( mon_etagere->CD[indice] );
}
//renvoie pointeur (constant) vers CD sur mon etagere
const CD_musique* cherche_CD_musique(const etagere* mon_etagere, int indice)
{
    return &( mon_etagere->CD[indice] );
}
```

```
struct CD_musique
{
    char titre[50];
    char auteur[50];
    int prix;
};
struct DVD_film
{
    char titre[50];
    int duree;
    int prix;
};
struct etagere
{
    struct CD_musique CD[100];
    struct DVD_film DVD[100];
};
```

```
int main()
{
    struct etagere mon_etagere;
    //rempli etagere ...

    struct CD_musique *mon_CD_1=retourne_CD_musique(&mon_etagere,3);
    mon_CD_1->prix=20; ← modifiable

    const struct CD_musique *mon_CD_2=cherche_CD_musique(&mon_etagere,3);
    printf("%d\n",mon_CD_2->prix); ← constant
}
```

# Mot clé **const**: Synthèse

- Passer les structs par pointeurs  
(choix)

- Tous pointeurs passés en **const** !  
(bonne programmation)

=> Enlever le const uniquement si nécessaire

```
char *filename="blabla";  
const char* filename="blabla";  
  
void ma_fonction(char *filename);  
void ma_fonction(const char *filename);
```

*vérification du compilateur  
code n'est pas impacté  
(aucun désavantage)*

# Mot clé **const**: placement

Identique:

```
const int a;  
int const a;
```

```
const int *a;  
int const *a;
```

Attention au placement de \*

```
int const *a;  
int *const a;
```

ne sont pas équivalents!

# Mot clé **const**: placement

- pointeur lui même constant → `char* const filename_1="fichier_1";`
- pointeur vers valeur constante → `const char* filename_2="fichier_2";`

```
char* const filename_1="fichier_1";  
const char* filename_2="fichier_2";  
  
filename_1=NULL; //erreur  
filename_2=NULL; //OK
```

- on peut cumuler: `const char* const filename_3="mon_fichier";`

- pointeurs multiples:

```
int **p1;  
int const **p2;  
int const *const *p3;  
int const *const *const p4;  
int *const *const p5;  
int **const p6;  
int *const*p7;
```

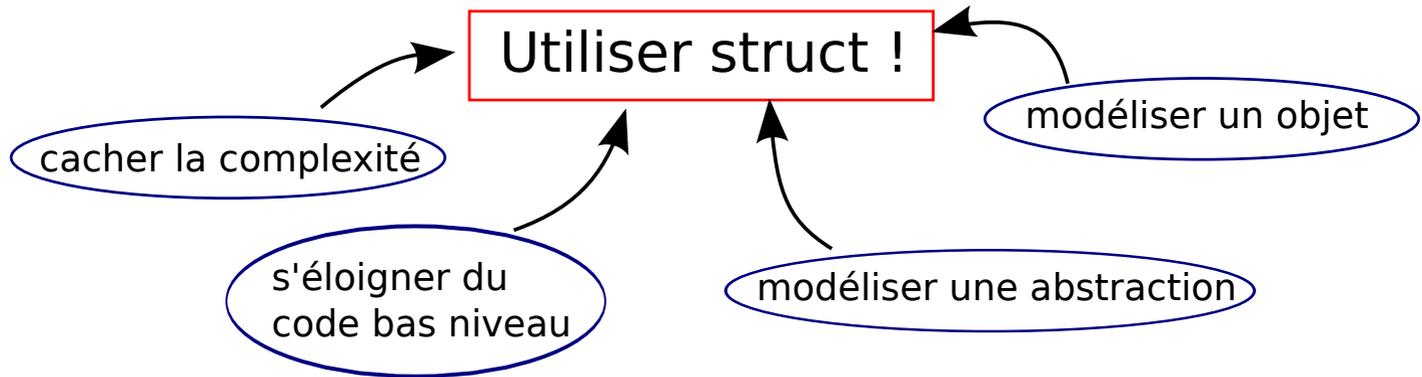
=> Trop complexe  
=> Utiliser des structs  
intermédiaires

# **Code de haut niveau:**

## *Modélisation d'abstraction*

Intérêt: notion d'abstraction  
Dénomination et structure  
Notion d'encapsulation

# Intérêt des structs



+ fonctions manipulants struct

# Intéret structs: Lisibilité

Préférer:

```
struct vecteur
{
    float x;
    float y;
};
float norme(const vecteur* v)
{
    return sqrt(v->x*v->x+v->y*v->y);
}
int main()
{
    struct vecteur mon_vecteur={1,2};
    float n=norme(&mon_vecteur);
    printf("%f\n",n);
}
```

lisible  
on connait quelles  
règles suit l'objet

Plutôt que:

```
int main()
{
    float v[2]={1,2};
    float n=sqrt(v[0]*v[0]+v[1]*v[1]);
    printf("%f\n",n);
}
```

On doit décrypter

# Intéret structs: Lisibilité



ex. Généricité

```
int main()
{
    struct vecteur mon_vecteur_1={1,2};
    struct vecteur mon_vecteur_2={4,5};
    struct vecteur mon_vecteur_3={5,-1};

    float n1=norme(&mon_vecteur_1);
    float n2=norme(&mon_vecteur_2);
    float n3=norme(&mon_vecteur_3);
}
```

OK

```
int main()
{
    float v1[2]={1,2};
    float v2[2]={4,5};
    float v3[2]={5,-1};

    float n1=sqrt(v1[0]*v1[0]+v1[1]*v1[1]);
    float n2=sqrt(v2[0]*v2[0]+v2[1]*v2[1]);
    float n3=sqrt(v3[0]*v3[0]+v3[1]*v3[1]);
}
```

A NE PAS FAIRE

=> Changez la norme 2 vers une norme infinie

```
float norme(const vecteur* v)
{
    return max(v->x, v->y);
}
```

1 ligne modifiée, local  
structure identique  
=>Modulaire



N lignes à modifier  
modification globale (tous le pgm)  
=> oublis, bugs, perte de temps



# But d'une struct

- Vos structs doivent cacher l'implémentation

mettre en avant l'objet manipulé

# Code de haut niveau:

## *Modélisation d'abstraction*

Intérêt: notion d'abstraction

→ **Dénomination et structure**

Notion d'encapsulation

# Structs: Dénomination

## Bonnes pratiques: Dénomination

OK: haut niveau

```
enum type_carburant {gasoil, essence, sans_plomb, GPL};  
struct voiture  
{  
    int immatriculation;  
    int kilometrage;  
    enum type_carburant carburant;  
};
```

+ A faire

```
struct voiture v1;  
v1.carburant=sans_plomb;
```



Trop proche du code

```
struct int_triplet  
{  
    int u_nbr; //immatriculation  
    int u_km; //kilometrage  
    int u_c; //carburant (0-gasoil,  
            //                1-essence,  
            //                2-sans_plomb,  
            //                3-GPL)  
};
```

mélange: int / voiture  
=>niveau d'abstraction non homogène

- A ne pas faire

```
struct int_triplet v2;  
v2.u_c=2;
```



# Structs: Structure

```
#define N 50

struct arbre
{
    int hauteur_tronc;
    char couleur_tronc[N];
    int largeur_tronc;
    int nombre_feuille;
    char couleur_feuille[N];
    int profondeur_racines;
};
```

- inhomogène

×

```
#define N 50
struct tronc_arbre
{
    int hauteur;
    char couleur[N];
    int largeur;
};
struct feuille_arbre
{
    int nombre;
    char couleur[N];
};
struct racine_arbre
{
    int profondeur;
};
```

```
struct arbre
{
    struct tronc_arbre tronc;
    struct feuille_arbre feuille;
    struct racine_arbre racine;
};
```

+ homogène  
=> Modulaire

✓

# Structs: Structure

```
struct nombre
{
    int n;
};
struct hauteur
{
    struct nombre h;
};
struct largeur
{
    struct nombre l;
};
struct nom
{
    char valeur[N];
};
struct couleur
{
    struct nom;
};

struct tronc_arbre
{
    struct hauteur h;
    struct largeur l;
    struct couleur c;
};
```

✗ excès inverse!

n'apporte pas d'information

Le niveau de hierarchie dépend de  
la complexité du modèle  
(taille du projet)

# Structs: Structure

Niveaux d'abstraction inhomogène!

```
enum type_arbre {epicea,chataignier,chene,eucalyptus};  
  
struct arbre  
{  
    int largeur;  
    int hauteur;  
    enum type_arbre type;  
    FILE* fid_disque;  
};
```

Mélange:  
détail d'implémentation  
/caractéristiques haut niveau

X

*caractéristiques  
haut niveau*

*descripteur de fichier  
niveau système:*

# Structs: Nombre de paramètres

```
struct arbre
{
    int largeur;
    int hauteur;
    enum type_arbre;
    int nombre_embranchement;
    int circonference;
    int poids;
    int profondeur_sous_sol;
    int nombre_fleurs;
    int flux_seve;
    int mois_fleurissement;
    int nombre_jour_fleurissement;
    int temperature_maximale;
    int quantitee_eau;
    int valeur_marche;
    int resistance_vent;
};
```

Trop de paramètres  
Mémoire humaine limitée

En moyenne:  
3-6 paramètres

**A NE PAS FAIRE!**

# Structs: Bonnes pratiques

## Synthèse

Une bonne struct:



Encapsule des données

Modélise un objet/une abstraction

Contient des paramètres homogènes

*Une mauvaise struct:*

*N'apporte pas d'information*

*Contient des informations sans coherences, ne modélise rien*

*Ne sert que de conteneur de variables au niveau C*

*Contient des noms peu significatifs*

# Code de haut niveau:

## *Modélisation d'abstraction*

Intérêt: notion d'abstraction  
Dénomination et structure

→ **Notion d'encapsulation**

# Struct: Notion d'abstraction

## struct + fonctions



Manipulation de données complexes  
Encapsulation

Ex.

Abstraction sphere:

- Comment est codé sphère ?
- Est-ce important pour l'utiliser ?
- Est-ce facile à lire ?

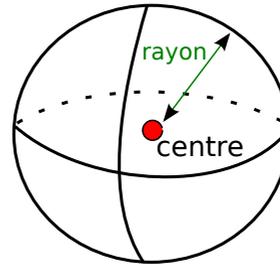
```
struct sphere
```

```
void sphere_init(struct sphere* s, float centre_x,  
                float centre_y,  
                float centre_z,  
                float rayon);  
  
float sphere_volume(const struct sphere* s);  
  
int main()  
{  
    struct sphere s1; sphere_init(&s1,0,0,0,1.0);  
    struct sphere s2; sphere_init(&s2,4,-5,6,2.0);  
  
    float volume_1=sphere_volume(&s1);  
    float volume_2=sphere_volume(&s2);  
}
```

# Struct: Notion d'abstraction

## Implémentation 1:

```
struct sphere
{
    float cx,cy,cz; //centre
    float R; //rayon
};
```



```
struct sphere
void sphere_init(struct sphere* s, float centre_x,
                float centre_y,
                float centre_z,
                float rayon);

float sphere_volume(const struct sphere* s);

int main()
{
    struct sphere s1; sphere_init(&s1,0,0,1.0);
    struct sphere s2; sphere_init(&s2,4,-5,0,2.0);

    float volume_1=sphere_volume(&s1);
    float volume_2=sphere_volume(&s2);
}
```

implémentation  
fonctions

```
void sphere_init(struct sphere* s, float centre_x,
                float centre_y,
                float centre_z,
                float rayon)
{
    assert(rayon>0);

    s->cx=centre_x;
    s->cy=centre_y;
    s->cz=centre_z;
    s->R=rayon;
}

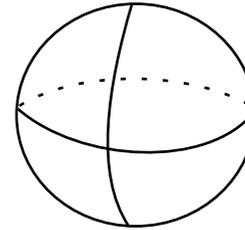
float sphere_volume(const struct sphere* s)
{
    assert(s->R>0);

    return 4.0/3.0*M_PI*(s->R*s->R*s->R);
}
```

# Struct: Notion d'abstraction

## Implémentation 2:

```
struct sphere
{
    //  $x^2 + ax + y^2 + by + z^2 + cz + d=0$ 
    float a,b,c,d;
};
```



$$x^2 + ax + y^2 + by + z^2 + cz + d = 0$$

implémentation  
fonctions

```
void sphere_init(struct sphere* s, float centre_x,
                float centre_y,
                float centre_z,
                float rayon)
{
    assert(rayon>0);

    s->a=-2*centre_x;
    s->b=-2*centre_y;
    s->c=-2*centre_z;
    s->d=centre_x*centre_x+centre_y*centre_y+centre_z*centre_z- rayon*rayon;
}

float sphere_volume(const struct sphere* s)
{
    float x0=-s->a/2.0;
    float y0=-s->b/2.0;
    float z0=-s->c/2.0;
    float R=sqrt(x0*x0+y0*y0+z0*z0-s->d);

    assert(R>0);

    return 4.0/3.0*M_PI*(R*R*R);
}
```

```
struct sphere
void sphere_init(struct sphere* s, float centre_x,
                float centre_y,
                float centre_z,
                float rayon);

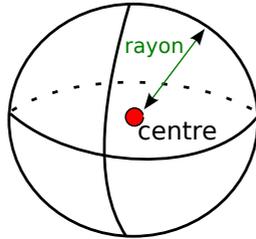
float sphere_volume(const struct sphere* s);

int main()
{
    struct sphere s1; sphere_init(&s1,0,0,0,1.0);
    struct sphere s2; sphere_init(&s2,4,-5,6,2.0);

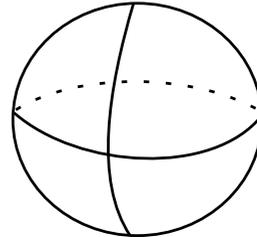
    float volume_1=sphere_volume(&s1);
    float volume_2=sphere_volume(&s2);
}
```

# Struct: Notion d'abstraction

Encapsulation:



VS



$$x^2+ax+y^2+by+z^2+cz+d=0$$

Peu importe l'implémentation  
=> **L'utilisation est la même**

L'interface (en tête) reste constante



```
struct sphere
```

```
void sphere_init(struct sphere* s, float centre_x,  
                float centre_y,  
                float centre_z,  
                float rayon);
```

```
float sphere_volume(const struct sphere* s);
```

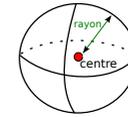
```
int main()
```

```
{  
  struct sphere s1; sphere_init(&s1,0,0,0,1.0);  
  struct sphere s2; sphere_init(&s2,4,-5,6,2.0);
```

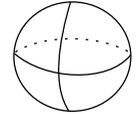
```
  float volume_1=sphere_volume(&s1);  
  float volume_2=sphere_volume(&s2);
```

```
}
```

# Struct: Modélisation d'abstraction

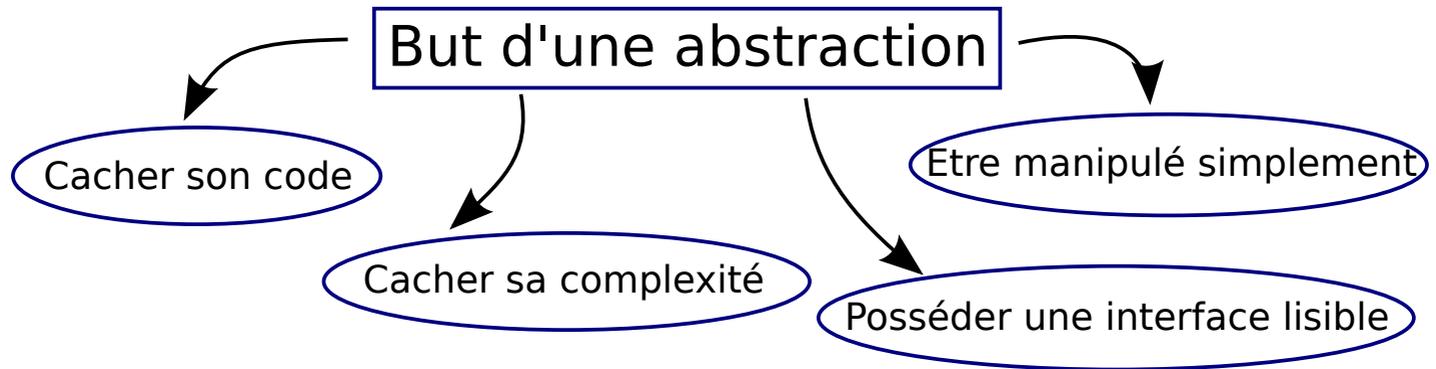


VS



$$x^2+ax+y^2+by+z^2+cz+d=0$$

Encapsulation:



Bonne encapsulation:

**Maintenable**  
**Evolutif**

*optimisation*  
*spécificité OS*  
*améliorations*  
...

# Struct: Exemple d'abstraction



## A ne pas faire:

```
int main()
{
    float x1=0,y1=0,z1=0,x2=4,y2=-5,z2=6;
    float R1=1,R2=2;

    float volume_1=4.0/3*M_PI*R1*R1*R1;
    float volume_2=4.0/3*M_PI*R2*R2*R2;
}
```

- pas lisible:  
=> abstraction sphere n'apparait pas
- pas d'évolution possible:  
=> changer implémentation=  
réécrire totalement le code

---

## A faire:

```
int main()
{
    struct sphere s1; sphere_init(&s1,0,0,0,1.0);
    struct sphere s2; sphere_init(&s2,4,-5,6,2.0);

    float volume_1=sphere_volume(&s1);
    float volume_2=sphere_volume(&s2);
}
```

- + lisible
- + évolutif

# Notion d'interface

Dans un logiciel:

- Celui qui lit/connait l'implémentation d'une fonction  
*1-3 personne*
- Celui qui utilise l'**interface** d'une fonction  
*ensemble des developpeurs*



le + utilisé  
le + important  
le + sensible  
le + de reflexion

# Bonnes pratiques de codage

Syntaxe tableau/pointeurs

Initialisation des variables

Dénomination des variables/fonctions

# Syntaxe pointeur/tableau

## Tableau

```
int main()
{
    int tableau[TAILLE];

    //accès a la case indice 5
    int a=tableau[5];
    //affectation sur la première case
    tableau[0]=7;
}
```

## Pointeur

```
int main()
{
    int* pointeur=NULL;
    int a=4;

    //pointe sur l'adresse de a
    pointeur=&a;

    //valeur du pointeur
    int b=*pointeur;

    //affectation de la valeur du pointeur
    *pointeur=8;
}
```

# Syntaxe pointeur/tableau

Tableau

T[k]

Pointeur

\*p

En C, tableau et pointeurs sont confondus

*Uniquement en C!*

**A NE PAS FAIRE:**

```
int main() X
{
    int tableau[TAILLE];
    int *p=tableau;

    int a=*(p+5);
    *(tableau+0)=7;
}
```

tableau avec  
syntaxe pointeur

```
int main() X
{
    int* pointeur=NULL;
    int a=4;
    pointeur=&a;
    int b=pointeur[0];
    pointeur[0]=8;
}
```

pointeur avec  
syntaxe tableau

# Syntaxe pointeur/tableau

Bonne pratique:

Interdisez vous:  $*(\text{pointeur}+k)$  ✗

Autodocumentation: Tableau : T[k]  
Pointeur : \*p ✓

Banissez: pointeurs multiples (int \*\*p) ✗

structurez en abstraction ✓

# Syntaxe pointeur/tableau

Exemples de portabilité (différents langages):

Vecteur en C++

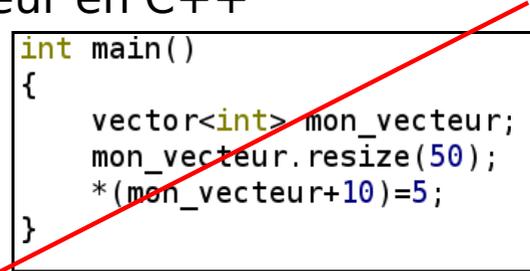
<pre>int main() {     vector&lt;int&gt; mon_vecteur;     mon_vecteur.resize(50);     mon_vecteur[10]=5; }</pre> 	<pre>int main() {     vector&lt;int&gt; mon_vecteur;     mon_vecteur.resize(50);     *(mon_vecteur+10)=5; }</pre> 
---	--

Tableau en Python

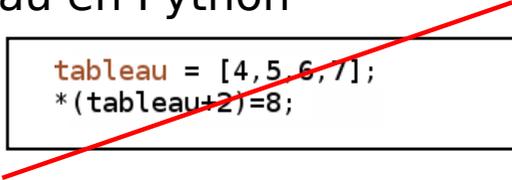
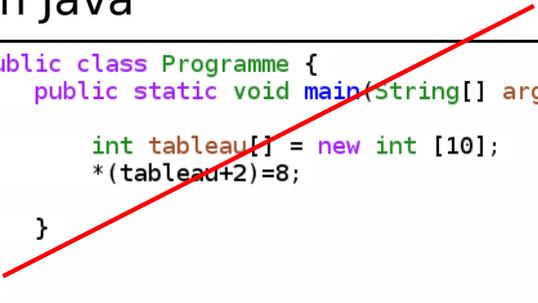
<pre>tableau = [4,5,6,7]; tableau[2]=8;</pre> 	<pre>tableau = [4,5,6,7]; *(tableau+2)=8;</pre> 
---	--

Tableau en Java

<pre>public class Programme {     public static void main(String[] args) {          int tableau[] = new int [10];         tableau[2]=8;      } }</pre> 	<pre>public class Programme {     public static void main(String[] args) {          int tableau[] = new int [10];         *(tableau+2)=8;      } }</pre> 
--	--

# Syntaxe pointeur/tableau

Ex. Listing de notes

Mathieu: {12,13,14,11,09,12}  
Francois: {5,7,11,11,10,8}  
Florence: {15,15,16,12,11,18}

```
int main()
{
    int T[3][6]={{12,13,14,11,9,12},
                {5,7,11,11,10,8},
                {15,15,16,12,11,18}};

    int **p=T;

    //acces a la 4eme note de mathieu:
    int note=*(p+3);
    //écriture de la 3eme note de Francois avec un 8
    (*(p+1)+2)=8;
}
```



Ecriture pointeur  
A ne pas faire

```
int main()
{
    int T[3][6]={{12,13,14,11,9,12},
                {5,7,11,11,10,8},
                {15,15,16,12,11,18}};

    //acces a la 4eme note de mathieu:
    int note=T[0][3];
    //écriture de la 3eme note de Francois avec un 8
    T[1][2]=8;
}
```

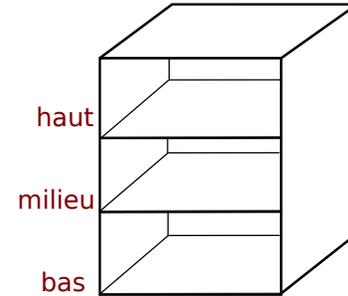
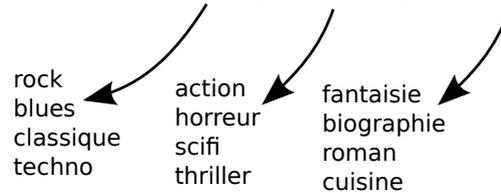


Ecriture tableau  
OK, avec documentation

# Syntaxe pointeur/tableau



## Ex. Etagère de CD/DVD/Livres



=> Enregistrement du nom de l'auteur/chanteur

Formalisme pointeur **X**  
(proche du code)

```
int main()
{
    char etagere[3][4][3][10][50];
    char *****p;

    //accès: chanteur du 4eme CD de rock sur l'etage du bas
    const char* chanteur=*(**(p+1)+3);

    //modification: auteur du 8eme livre de cuisine sur l'etage du haut
    strcpy( *((*(p+2)+2)+7), "Maite et Micheline");
}
```

court  
lisible?  
debug-able?  
correct?

Formalisme bibliotheque **✓✓**  
(proche de l'objet reel)

```
#define NOMBRE_TYPE_OBJET 3
#define NOMBRE_CATEGORIE 4
#define NOMBRE_ETAGE 3
#define NOMBRE_MAX_OBJET 10
#define TAILLE_MAX_NOM_AUTEUR 50

enum type_objet {DVD, CD, livre};
enum type_etage {bas, milieu, haut};

enum type_DVD {action, horreur, scifi, triller};
enum type_CD {rock, blues, classique, techno};
enum type_livre {fantaisie, biographie, roman, cuisine};

int main()
{
    char etagere[NOMBRE_TYPE_OBJET]
                [NOMBRE_CATEGORIE]
                [NOMBRE_ETAGE]
                [NOMBRE_MAX_OBJET]
                [TAILLE_MAX_NOM_AUTEUR];

    //accès: chanteur du 4eme CD de rock sur l'etage du bas
    const char *chanteur=etagere[CD][rock][bas][3];

    //modification: auteur du 8eme livre de cuisine sur l'etage du haut
    strcpy(etagere[livre][cuisine][haut][7], "Maite et Micheline");
}
```

commentaire quasi-inutile

# Bonnes pratiques de codage

Syntaxe tableau/pointeurs

→ **Initialisation des variables**

Dénomination des variables/fonctions

# Initialisation des Variables

Bonne pratique: règle à appliquer

**Toujours**

**Toujours** initialiser ses variables

**Toujours**

**Toujours**

**Toujours**



# Initialisation des Variables

## Exemple:

```
//entiers
int entier_1=0;
int entier_2=-1;
unsigned long long int entier_3=0;
```

généralement  
entiers à 0 (ou -1)

```
//caractere
char caractere='0';
```

```
//flottants
float flottant_1=0.0;
double flottant_2=0.0;
```

généralement  
flottants à 0

```
//tableaux statiques
int tableau_1[5]={0,0,0,0,0};
char tableau_2[5]='\0','\0','\0','\0','\0'; \0: fin de chaine
float tableau_3[5]={0.0,0.0,0.0,0.0,0.0};
```

```
//grands tableaux
int tableau_5[5672];memset(tableau_5,0,5672);
int tableau_4[5672]={[0 ... 5671] 1}; //gcc (C99) uniquement
```

```
//pointeurs
void* pointeur_1=NULL;
const char* pointeur_2=NULL;
int* pointeur_3=NULL;
int*** pointeur_4=NULL;
```

Pointeurs toujours à NULL  
(ou adresse finale)  
**Jamais** de pointeurs non initialisés!!!

# Initialisation des Variables

Exemple:

```
int entier;  
printf("%d\n",entier);  
  
float flottant;  
printf("%f\n",flottant);
```



```
32767  
379680036110150551298113536.000000
```

Dépend du:

*ystème*

*compilateur*

*code*

*état de la mémoire (autres programmes)*

=> **comportement indéterminé**

# Initialisation des Structs

Les structs doivent avoir leur fonction d'initialisation



Exemple:

```
#define TAILLE_NOM_MAX 60

struct livre
{
    char auteur[TAILLE_NOM_MAX];
    char titre[TAILLE_NOM_MAX];
    int nombre_de_page;
};
```

Pas d'initialisation: X

```
void livre_affiche(const struct livre* livre_a_afficher)
{
    printf("auteur: %s\n", livre_a_afficher->auteur);
    printf("titre: %s\n", livre_a_afficher->titre);
    printf("nbr pages: %d\n", livre_a_afficher->nombre_de_page);
}

int main()
{
    struct livre mon_livre;
    livre_affiche(&mon_livre);
}
```

```
auteur: 00
         01
titre: ; 00
         7F
nbr pages: 4195073
```

# Initialisation des Structs

Les structs doivent avoir leur fonction d'initialisation

Exemple:

```
#define TAILLE_NOM_MAX 60  
  
struct livre  
{  
    char auteur[TAILLE_NOM_MAX];  
    char titre[TAILLE_NOM_MAX];  
    int nombre_de_page;  
};
```

Fonction d'initialisation: ✓

```
void livre_init(struct livre* livre_a_initialiser)  
{  
    strcpy(livre_a_initialiser->auteur, "Auteur Inconnu");  
    strcpy(livre_a_initialiser->titre, "Titre Inconnu");  
    livre_a_initialiser->nombre_de_page=-1;  
}
```

```
int main()  
{  
    struct livre mon livre;  
    livre_init(&mon_livre);  
    livre_affiche(&mon_livre);  
}
```

```
auteur: Auteur Inconnu  
titre: Titre Inconnu  
nbr pages: -1
```

# Initialisation des Structs

Exemple de cas d'erreur:

```
int main()  
{  
    struct livre ensemble_livre[3];  
  
    strcpy(ensemble_livre[0].auteur, "Jules Verne");  
    strcpy(ensemble_livre[0].titre, "20000 lieux sous les mers");  
    ensemble_livre[0].nombre_de_page=200;  
  
    strcpy(ensemble_livre[2].auteur, "H.G. Wells");  
    strcpy(ensemble_livre[2].titre, "When the Sleeper Wakes");  
    ensemble_livre[2].nombre_de_page=150;  
  
    int k=0;  
    int nombre_total_pages=0;  
    for(k=0;k<3;++k)  
        nombre_total_pages += ensemble_livre[k].nombre_de_page;  
    printf("%d\n", nombre_total_pages);  
}
```



Affichage: 3117

OK? Erreur? Debug?

# Initialisation des Structs

Exemple de cas d'erreur:

```
int main()
{
    struct livre ensemble_livre[3];
    int k=0;
    for(k=0;k<3;++k)
        livre_init(&ensemble_livre[k]);

    strcpy(ensemble_livre[0].auteur,"Jules Verne");
    strcpy(ensemble_livre[0].titre,"20000 lieux sous les mers");
    ensemble_livre[0].nombre_de_page=200;

    strcpy(ensemble_livre[2].auteur,"H.G. Wells");
    strcpy(ensemble_livre[2].titre,"When the Sleeper Wakes");
    ensemble_livre[2].nombre_de_page=150;

    int nombre_total_pages=0;
    for(k=0;k<3;++k)
    {
        int nbr=ensemble_livre[k].nombre_de_page;
        if(nbr!=-1)
            nombre_total_pages += ensemble_livre[k].nombre_de_page;
        else
            {printf("Erreur ensemble_livre[%d] invalide\n",k);exit(1);}
    }
    printf("%d\n",nombre_total_pages);
}
```



Affichage: *Erreur ensemble\_livre[1] invalide*

=> **Debug aisé**

# Initialisation des Structs

## Exemple 2

```
struct v3
{
    float x;
    float y;
    float z;
};

void v3_init(struct v3* vec)
{
    vec->x=0;
    vec->y=0;
    vec->z=0;
}

int main()
{
    struct v3 vec_1; v3_init(&vec_1);
    struct v3 vec_2; v3_init(&vec_2);

    vec_2.x = 5;
    vec_1.y += vec_2.x;
}
```



Avantages:

Detection d'erreurs  
Code plus sure  
Portabilité  
Répétabilité

Bonne pratique:

Pour toute struct => fonction d'initialisation

Pour toute déclaration => appel à l'initialisation

Initialisation a des valeurs caracteristiques interessantes  
(detection d'erreur, valeur nulle, ...)

# Initialisation des Structs



## Optimisation ?

```
int main()
{
    int a=0;

    int k=-1;
    for(k=0;k<10;++k)
    {
        a += 5*k+1;
        printf("%d\n", a);
    }
}
```

avec  
initialisation

```
int main()
{
    int a=0;

    int k;
    for(k=0;k<10;++k)
    {
        a += 5*k+1;
        printf("%d\n", a);
    }
}
```

sans  
initialisation

\$ gcc -O2

Code assembleur  
identique

```
main:
.LFB0:
    .cfi_startproc
    pushq   %rbp
    .cfi_def_cfa_offset 16
    .cfi_offset 6, -16
    xorl   %ebp, %ebp
    pushq  %rbx
    .cfi_def_cfa_offset 24
    .cfi_offset 3, -24
    movl   $1, %ebx
    subq   $8, %rsp
    .cfi_def_cfa_offset 32
    .p2align 4,,10
    .p2align 3
```

Il n'y a aucune gain de  
performance à ne pas initialiser!

# Initialisation des Structs



## Optimisation ?

### Vecteur 3D d'entiers

```
struct v3
{
    int x,y,z;
};

void v3_init(struct v3* vec)
{
    vec->x=0;vec->y=0;vec->z=0;
}

void v3_affecte(struct v3* vec,float x,float y,float z)
{
    vec->x=x;vec->y=y;vec->z=z;
}

void v3_affiche(const struct v3* vec)
{
    printf("( %d,%d,%d)\n",vec->x,vec->y,vec->z);
}
```

# Initialisation des Structs



## Optimisation ?

### Avec initialisation

```
int main()
{
    struct v3 vec; v3_init(&vec);
    v3_affecte(&vec, 4, 5, 6);
    v3_affiche(&vec);
}
```

### Sans initialisation

```
int main()
{
    struct v3 vec;
    v3_affecte(&vec, 4, 5, 6);
    v3_affiche(&vec);
}
```

Vecteur 3D d'entiers

```
struct v3
{
    int x, y, z;
};

void v3_init(struct v3* vec)
{
    vec->x=0; vec->y=0; vec->z=0;
}

void v3_affecte(struct v3* vec, float x, float y, float z)
{
    vec->x=x; vec->y=y; vec->z=z;
}

void v3_affiche(const struct v3* vec)
{
    printf("(%d,%d,%d)\n", vec->x, vec->y, vec->z);
}
```

### Assembleur identique

```
main:
.LFB3:
    .cfi_startproc
    subq    $24, %rsp
    .cfi_def_cfa_offset 32
    movq    %rsp, %rdi
    movl    $4, 4(%rsp)
    movl    $5, 4(%rsp)
    movl    $6, 8(%rsp)
    call   v3_affiche
    addq    $24, %rsp
    .cfi_def_cfa_offset 8
    ret
    .cfi_endproc
```

\$ gcc -O2

=> Performance identique

# Initialisation des Structs



## Optimisation ?

```
int main()
{
    struct v3 vec; v3_init(&vec);
    v3_affiche(&vec);
    v3_affecte(&vec, 4, 5, 6);
    v3_affiche(&vec);
}
```

initialisation

```
main:
.LFB3:
.cfi_startproc
subq   $24, %rsp
.cfi_def_cfa_offset 32
movq   %rsp, %rdi
movl   $0, (%rsp)
movl   $0, 4(%rsp)
movl   $0, 8(%rsp)
call   v3_affiche
movq   %rsp, %rdi
movl   $4, (%rsp)
movl   $5, 4(%rsp)
movl   $6, 8(%rsp)
call   v3_affiche
addq   $24, %rsp
.cfi_def_cfa_offset 8
ret
.cfi_endproc
```

affiche (0,0,0)

\$ gcc -O2

```
int main()
{
    struct v3 vec;
    v3_affiche(&vec);
    v3_affecte(&vec, 4, 5, 6);
    v3_affiche(&vec);
}
```

```
main:
.LFB3:
.cfi_startproc
subq   $24, %rsp
.cfi_def_cfa_offset 32
movq   %rsp, %rdi
call   v3_affiche
movq   %rsp, %rdi
movl   $4, (%rsp)
movl   $5, 4(%rsp)
movl   $6, 8(%rsp)
call   v3_affiche
addq   $24, %rsp
.cfi_def_cfa_offset 8
ret
.cfi_endproc
```

affichage incorrect

=> gcc réalise l'initialisation uniquement si cela est utile!

```
struct v3
{
    int x, y, z;
};

void v3_init(struct v3* vec)
{
    vec->x=0; vec->y=0; vec->z=0;
}

void v3_affecte(struct v3* vec, float x, float y, float z)
{
    vec->x=x; vec->y=y; vec->z=z;
}

void v3_affiche(const struct v3* vec)
{
    printf("%d,%d,%d\n", vec->x, vec->y, vec->z);
}
```

Conclusion:

**Toujours** initialiser dans le code !!!

*les built-in  
et les structs !*

Code + sécurisé

Debug + aisé

Pas de perte de performances!

# Bonnes pratiques de codage

Syntaxe tableau/pointeurs

Initialisation des variables

→ **Dénomination des variables/fonctions**

# Dénomination: Structs + Fonctions

En C: Nom de fonction doit être unique  
*Pour l'ensemble du programme !*

Rendez votre nom unique et précis



Le nom doit indiquer  
- **ce que fait** la fonction  
- **sur quoi** elle agit

*nom d'une fonction  
= documentation*

Suivez une règle de noms/passage arguments cohérente  
tout au long du programme

# Dénomination: Structs + Fonctions

## Exemple

```
struct arbre;  
struct voiture;  
  
void voiture_vend(struct voiture* v, int prix_vente);
```

fonction agissant  
sur struct voiture

on retrouve l'entité voiture

nom explicite

dénomination unique et explicite: vente de l'objet voiture

# Dénomination: Structs + Fonctions

Exemple

```
struct arbre;  
struct voiture;  
  
void voiture_vend(struct voiture* v, int prix_vente);
```



fonction agissant sur struct voiture  
on retrouve l'entitée voiture  
nom explicite  
dénomination unique et explicite: vente de l'objet voiture

```
struct arbre;  
struct voiture;  
  
void f5(struct voiture* v, int p);
```



mauvaise dénomination: aucune information variable non explicite

```
struct arbre;  
struct voiture;  
  
float longueur();
```



manque précision: longueur de quelle entitée?

```
struct Arbre;  
struct voiture;  
  
void VoitureVend(struct voiture* v, int Prix_Vente);  
void Arbre_Coupe(struct Arbre a);
```



Manque cohérence dénomination. => Suivez une seule règle unique (majuscule, underscore, ...)

```
struct arbre;  
struct voiture;  
  
void voiture_vente(struct voiture* v, int prix_vente);  
void voiture_achete(int prix_achat, struct voiture* v);  
void deplace(struct voiture* v, const char* destination);
```



manque cohérence argument 1ere/dernière position indication entitée manquante

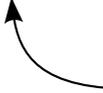
# Dénomination: Structs + Fonctions

## Proposition de dénomination

Fonction agissant sur une struct:

```
type_retour nom_struct_action((const) struct entitee* nom, types autres_parametres ...)
```

struct sur laquelle  
agit la fonction en 1ère place



exemple:

```
int voiture_prix(const struct voiture* v);  
  
struct etudiant* classe_recherche_etudiant(struct classe* classe_courante,  
                                           const char* nom_etudiant);  
  
void arbre_initialise(struct arbre* a, int longueur, int hauteur);  
  
void couleur_cmjn_vers_rgb(const struct couleur_cmjn* origine,  
                          struct couleur_rgb* destination);
```

=> une partie de la documentation donnée par les noms

# Dénomination: Structs + Fonctions

## Proposition de dénomination

Fonction retour booleen: retour = vrai(1)/faux(0)

type\_retour **nom\_struct\_est\_qualificatif**(const struct\* entitee)

ou

type\_retour **is\_nom\_struct\_qualificatif**(const struct\* entitee)

```
int arbre_est_vivant(const struct arbre* a);  
int voiture_est_prete(const struct voiture* v);  
int nombre_est_pair(int nombre);
```

```
int is_tree_alive(const struct tree* t);  
int is_window_open(const struct window* w);  
int is_pointer_null(const void* pointer);
```

# Dénomination: Structs + Fonctions

## Proposition de dénomination

Fonction retour booleen: retour = vrai(1)/faux(0)

type\_retour **nom\_struct\_est\_qualificatif**(const struct\* entitee)

ou

type\_retour **is\_nom\_struct\_qualificatif**(const struct\* entitee)

Note: Possibilité d'émuler variable booléenne en C

```
typedef int boolean;
#define VRAI 1
#define FAUX 0

boolean arbre_est_vivant(const struct arbre* a);

int main()
{
    struct arbre a;
    arbre_initialise(&a);

    while( arbre_est_vivant==VRAI )
    {
        arbre_grandit(&a);

        boolean est_malade=arbre_est_malade();
        if(est_malade==VRAI)
            printf("Arbre malade\n");
    }

    return 0;
}
```



# Dénomination: Variables



Nommez vos variables avec soin

lisible

compréhensible

précis

Bonne pratique:

Nom de variable entre 6-15 caractères

# Dénomination: Variables



Bonne pratique:

Donnez du sens à vos noms de variables

```
#define N 15
int tableau[N];
int limite_1=3;
int limite_2=10;

int fonction_1(int x)
{
    if(x>limite_1)
    {
        if(x>limite_2)
            fonction_2();
        else
            fonction_3();
    }
    else
    {
        printf("Elimine\n");
        abort();
    }
}
```



de quoi parle-on?

```
#define NOMBRE_ETUDIANT 15
int note_ensemble_etudiants[MAX_ETUDIANT];
int note_elimatoire=3;
int note_passage=10;

int recoit_note(int note_etudiant,int id_etudiant)
{
    if(note_etudiant>note_elimatoire)
    {
        if(note_etudiant>note_passage)
        {
            note_ensemble_etudiants[id_etudiant]=note_etudiant;
            validation_matiere();
        }
        else
            seconde_session();
    }
    else
    {
        printf("Elimine\n");
        abort();
    }
}
```



OK

# Dénomination: Variables



## Bonne pratique:

+ portée d'une variable est grande, plus le nom doit être précis.

```
#define NOMBRE_MAX_ETUDIANTS 160
#define NOMBRE_MAX_ENSEIGNANTS 5

#define TAILLE_MAX_NOM 20
struct etudiant
{
    char nom[TAILLE_MAX_NOM];
    int note;
};
struct classe
{struct etudiant classe[NOMBRE_MAX_ETUDIANTS];};

char nom_fichier_sauvegarde_etudiants[]="nom_etudiants.txt";

void classe_recupere_note(const struct classe* classe_courante,
    const char* nom_etudiant_a_chercher);

int main()
{
    struct classe eti3;

    charge_nom_etudiants(&eti3,nom_fichier_sauvegarde_etudiants);

    char nom_a_chercher[]="Benjamin Dumont";
    int note_etudiant=classe_recupere_note(eti3,nom_a_chercher);

    if(note_etudiant<10)
        envoie_mail("Session 2\n");

    return 0;
}

int classe_recupere_note(const struct classe* c,const char* nom)
{
    int k=0;
    while(k<NOMBRE_MAX_ETUDIANTS)
    {
        int comp=strcmp(c->classe[k].nom,nom);
        if(comp==0)
            return c->classe[k].note;
        ++k;
    }
    printf("Etudiant %s non trouve\n",nom);
    return -1;
}
```

variables globales

variables locales

```
#define N1 160
#define N2 5

#define N3 20
struct e
{
    char n[N3];
    int x;
};
struct c
{
    struct e classe[N1];
};
char f[]="nom_etudiants.txt";

void classe_recupere_note(const struct classe* c,
    const char* n);

int main()
{
    struct classe classe_etudiant_eti3;

    charge_nom_etudiants(&classe_etudiant_eti3,f);

    char nom_a_chercher[]="Benjamin Dumont";
    int note_etudiant=classe_recupere_note(classe_etudiant_eti3,nom_a_chercher);

    if(note_etudiant<10)
        envoie_mail("Session 2\n");

    return 0;
}

int classe_recupere_note(const struct classe* classe_etudiant_courante,
    const char* nom_de_l_etudiant)
{
    int indice_de_parcours=0;
    while(indice_de_parcours<N1)
    {
        int retour_comparaison_chaine_caractere=strcmp(
            classe_etudiant_courante->classe[k].nom,
            nom_de_l_etudiant);
        if(retour_comparaison_chaine_caractere==0)
            return classe_etudiant_courante->classe[k].note;
        ++indice_de_parcours;
    }
    printf("Etudiant %s non trouve\n",nom);
    return -1;
}
```

peu compréhensible

inutilement complexe  
peu lisible

# Mémoire dynamique

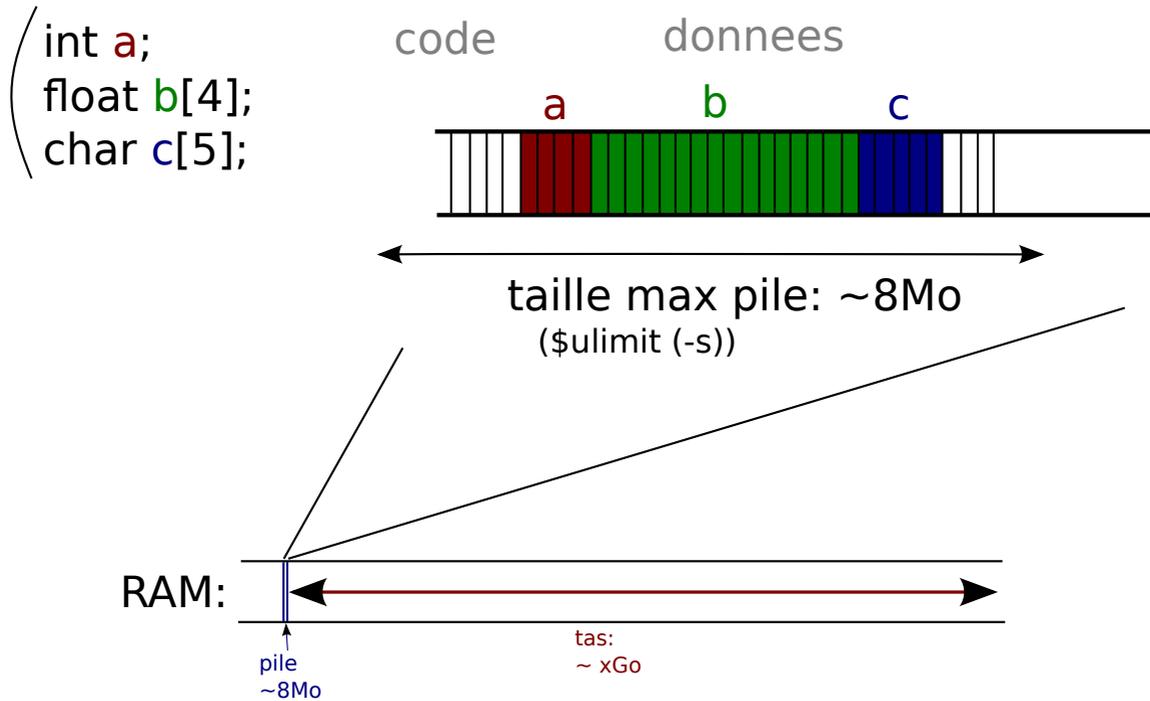
Allocation

Structure de données:

chainage dynamique et graphes

# Mémoire dynamique

Toutes les variables déclarées explicitement sont dans la pile (stack)



# Mémoire dynamique

Il est possible d'allouer des emplacements mémoire dans le tas  
(heap)

**Avantages:** + Peut gérer les grandes quantités de données  
+ Peut allouer des tableaux de tailles variables

**Inconvénient:** - N'est pas géré automatiquement en C



Allocation/occupation d'espace mémoire par:  
`malloc(taille)`

Désallocation/libération d'espace mémoire par:  
`free(adresse)`

=> A la charge du programmeur de bien gérer l'allocation/libération

=> **Attention: >90% des erreurs se font sur la gestion mémoire**

Note: Le C n'est pas le langage le + adapté pour la gestion de mémoire simple

# Mémoire dynamique



Exemple:

```
int main()
{
    int *mon_tableau=NULL;
    mon_tableau=malloc(5*sizeof(int));

    if(mon_tableau==NULL)
        {printf("Erreur allocation memoire\n");exit(1);}

    int k=0;
    for(k=0;k<5;++k)
        mon_tableau[k]=2*k;

    for(k=0;k<5;++k)
        printf("%d\n",mon_tableau[k]);

    free(mon_tableau);
    mon_tableau=NULL;

    return 0;
}
```

allocation espace mémoire

vérification

utilisation comme un tableau standard

libération mémoire

# Mémoire dynamique



Exemple:

```
int main()
{
    int *mon_tableau=NULL;
    mon_tableau=malloc(5*sizeof(int));
    if(mon_tableau==NULL)
        {printf("Erreur allocation memoire\n");exit(1);}

    int k=0;
    for(k=0;k<5;++k)
        mon_tableau[k]=2*k;

    for(k=0;k<5;++k)
        printf("%d\n",mon_tableau[k]);

    free(mon_tableau);
    mon_tableau=NULL;

    return 0;
}
```

erreur classique: malloc(5)  
oublie taille de l'entier=4 octets

erreur classique: oublie verification

erreur classique: dépassement tableau

erreur classique: oublie libération  
(perte d'espace RAM!)

# Mémoire dynamique



Exemple: tableau de taille non connue à la compilation

```
int main()
{
    printf("Donnez un nombre de cases a allouer positif: ");

    int n=0;
    scanf ("%d",&n);

    if(n<=0 || n>5000000)
    {
        printf("Nombre %d invalide\n",n);
        exit(1);
    }

    int *tableau=NULL;
    tableau=malloc(n*sizeof(int));

    printf("Je viens d'allouer dynamiquement un tableau de %d entier\n",n);

    int k=0;
    for(k=0;k<n;++k)
        tableau[k]=k;

    free(tableau);
    tableau=NULL;
}
```

# Mémoire dynamique



## Exemple: redimensionnement d'un tableau

```
int main()
{
    int taille_1=500;
    tableau=malloc(taille_1*sizeof(float));

    if(tableau==NULL)
        {printf("Erreur allocation tableau\n");exit(1);}

    int k=0;
    for(k=0;k<taille_1;++k)//remplissage de 500 cases
        tableau[k]=cos((float)k/taille_1*2*M_PI);

    int taille_2=1000;
    copie_et_aggrandissement_tableau(taille_2);

    //remplissage des 500 cases suivantes
    for(k=taille_1;k<taille_2;++k)
        tableau[k]=k*k;

    free(tableau);//liberation de l'espace memoire
    tableau=NULL;

    return 0;
}
```

```
float *tableau=NULL;

void copie_et_aggrandissement_tableau(int nouvelle_taille)
{
    float *tableau_temporaire=tableau;//copie du pointeur

    //allocation du tableau avec nouvelle taille
    tableau=NULL;
    tableau=malloc(nouvelle_taille);

    if(tableau==NULL)
        {printf("Erreur allocation tableau\n");exit(1);}

    //copie des valeurs du tableau precedent
    int k=0;
    for(k=0;k<nouvelle_taille;++k)
        tableau[k]=tableau_temporaire[k];

    //liberation du tableau precedent
    free(tableau_temporaire);
    tableau_temporaire=NULL;
}
```

# Mémoire dynamique

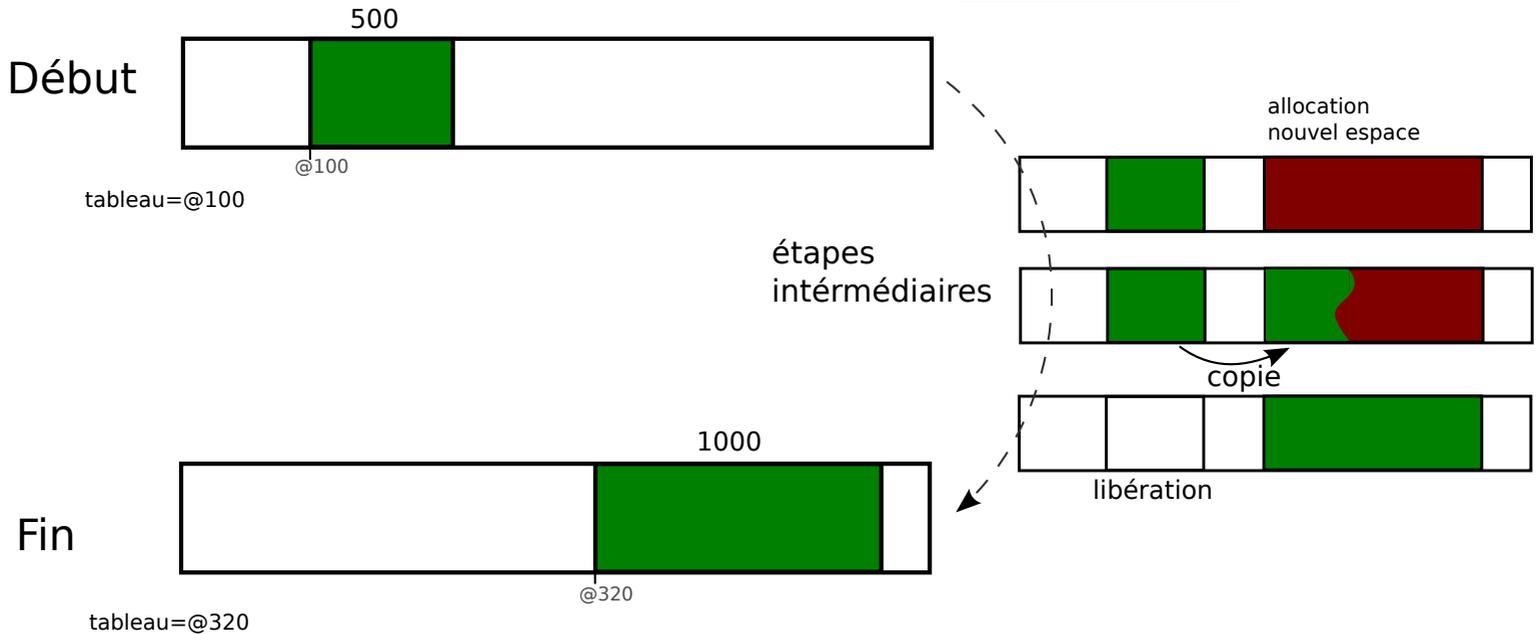


Exemple: redimensionnement d'un tableau

```
int main()
{
    int taille_1=500;
    tableau=malloc(taille_1*sizeof(float));
    if(tableau==NULL)
        printf("Erreur allocation tableau\n");exit(1);
    int i=0;
    for(k=0;k<taille_1;++k)//remplissage de 500 cases
        tableau[k]=cos((float)k/taille_1*2*M_PI);
    int taille_2=1000;
    copie_et_agrandissement_tableau(taille_2);
    //remplissage des 500 cases suivantes
    for(k=taille_1;k<taille_2;++k)
        tableau[k]=k;
    free(tableau);//liberation de l'espace memoire
    tableau=NULL;
    return 0;
}

float *tableau=NULL;
void copie_et_agrandissement_tableau(int nouvelle_taille)
{
    float *tableau_temporaire=tableau;//copie du pointeur
    //allocation du tableau avec nouvelle taille
    tableau=NULL;
    tableau=malloc(nouvelle_taille);
    if(tableau==NULL)
        printf("Erreur allocation tableau\n");exit(1);
    int i=0;
    //copie des valeurs du tableau precedent
    for(k=0;k<nouvelle_taille;++k)
        tableau[k]=tableau_temporaire[k];
    //liberation du tableau precedent
    free(tableau_temporaire);
    tableau_temporaire=NULL;
}
```

Principe:



**Attention:** Aggrandissement de tableau=  
nouvelle allocation+copie = long !

# Mémoire dynamique



Exemple: mémoire dynamique sur une struct

```
struct tronc
{
    float epaisseur_ecorce;
    int nombre_anneaux;
};

struct feuilles
{
    float largeur;
    float longueur;
};

struct arbre
{
    struct tronc;
    struct feuilles;
};

int main()
{
    struct arbre *foret=NULL;
    foret=malloc(3*sizeof(struct arbre));

    foret[0].tronc.epaisseur_ecorce=1.5;
    foret[1].feuilles.largeur=5.0;

    free(foret);
    foret=NULL;

    return 0;
}
```

# Mémoire dynamique

Allocation

→ **Structure de données:**  
chainage dynamique et graphes

# Mémoire dynamique



## Exemple: structure de données avancées: liste chaînées

```
struct maillon
{
    int valeur;
    struct maillon *suivant;
};

int main()
{
    struct maillon m;
    m.valeur=5;
    m.suivant=malloc(sizeof(struct maillon));

    struct maillon* m2=NULL;
    m2=m.suivant;
    m2->valeur=8;

    m2->suivant=malloc(sizeof(struct maillon));

    struct maillon* m3=NULL;
    m3=m2->suivant;
    m3->valeur=2;

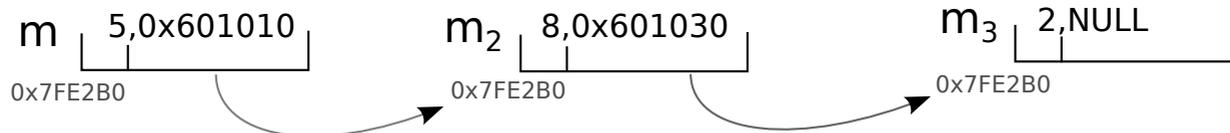
    printf("%d %d %d\n", m.valeur,
           m.suivant->valeur,
           m.suivant->suivant->valeur);

    printf("%d %d %d\n", m.valeur, m2->valeur, m3->valeur);

    return 0;
}
```

▼ m	
valeur	5
▼ → suivant	0x601010
▼ *0x601010	
valeur	8
▼ → suivant	0x601030
▼ *0x601030	
valeur	2
▼ → suivant	0x0
*0x0	Cannot access memory at address 0x0
▼ → m2	0x601010
▼ *0x601010	
valeur	8
▶ → suivant	0x601030
▼ → m3	0x601030
▼ *0x601030	
valeur	2
▶ → suivant	0x0

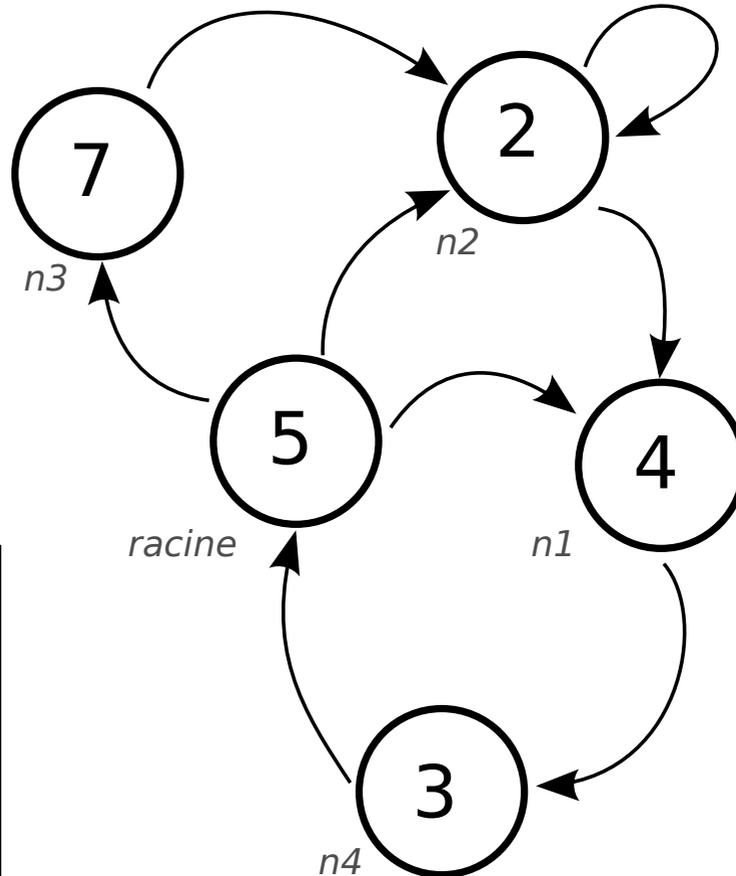
(tableau de taille quelconque  
qui peut d'agrandir)



# Mémoire dynamique



Exemple: structure de données avancées: graphe



```
struct noeud
{
  int valeur;

  int nombre_fils;
  struct noeud **fils;
};
```

# Mémoire dynamique



Exemple: structure de données avancées: graphe

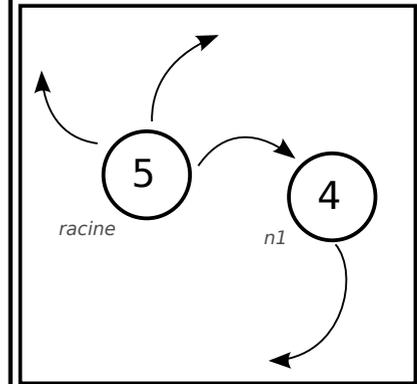
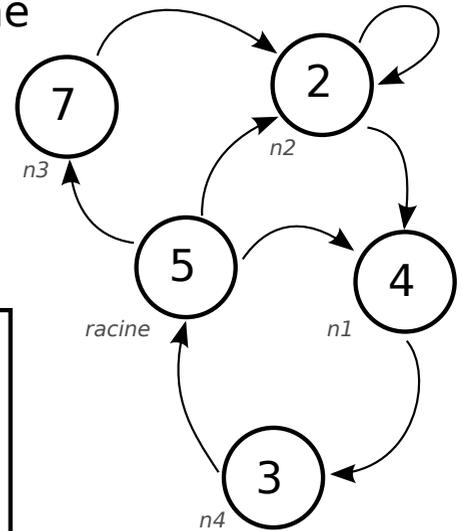
```
struct noeud
{
    int valeur;

    int nombre_fils;
    struct noeud **fils;
};
```

```
int main()
{
    //allocation
    struct noeud *racine=NULL;
    racine=malloc(sizeof(struct noeud));
    assert(racine!=NULL);

    racine->valeur=5;
    racine->nombre_fils=3;
    racine->fils=NULL;
    racine->fils=malloc(racine->nombre_fils*sizeof(struct noeud *));
    assert(racine->fils!=NULL);

    struct noeud* n1=NULL;
    n1=malloc(sizeof(struct noeud));
    assert(n1!=NULL);
    n1->valeur=4;
    racine->fils[0]=n1;
    n1->nombre_fils=1;
    n1->fils=NULL;
    n1->fils=malloc(n1->nombre_fils*sizeof(struct noeud *));
    assert(n1->fils!=NULL);
```



# Mémoire dynamique



Exemple: structure de données avancées: graphe

```
struct noeud
{
    int valeur;

    int nombre_fils;
    struct noeud **fils;
};
```

```
int main()
{
    //allocation
    struct noeud *racine=NULL;
    racine=malloc(sizeof(struct noeud));
    assert(racine!=NULL);

    racine->valeur=5;
    racine->nombre_fils=3;
    racine->fils=NULL;
    racine->fils=malloc(racine->nombre_fils*sizeof(struct noeud *));
    assert(racine->fils!=NULL);

    struct noeud* n1=NULL;
    n1=malloc(sizeof(struct noeud));
    assert(n1!=NULL);
    n1->valeur=4;
    racine->fils[0]=n1;
    n1->nombre_fils=1;
    n1->fils=NULL;
    n1->fils=malloc(n1->nombre_fils*sizeof(struct noeud *));
    assert(n1->fils!=NULL);
```

```
    struct noeud* n2=NULL;
    n2=malloc(sizeof(struct noeud));
    assert(n2!=NULL);
    n2->valeur=2;
    n2->nombre_fils=2;
    n2->fils=NULL;

    n2->fils=malloc(n2->nombre_fils*sizeof(struct noeud *));
    n2->fils[0]=n1;
    n2->fils[1]=n2;

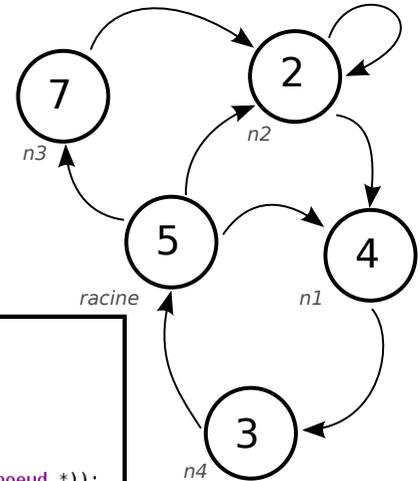
    racine->fils[1]=n2;

    struct noeud* n3=NULL;
    n3=malloc(sizeof(struct noeud));
    assert(n3!=NULL);
    n3->valeur=7;
    n3->nombre_fils=1;
    n3->fils=NULL;
    n3->fils=malloc(sizeof(struct noeud*));
    assert(n3->fils!=NULL);
    n3->fils[0]=n2;

    racine->fils[2]=n3;

    struct noeud* n4=NULL;
    n4=malloc(sizeof(struct noeud));
    assert(n4!=NULL);
    n4->valeur=3;
    n4->nombre_fils=1;
    n4->fils=NULL;
    n4->fils=malloc(n4->nombre_fils*sizeof(struct noeud*));
    assert(n4->fils!=NULL);
    n4->fils[0]=racine;

    n1->fils[0]=n4;
```

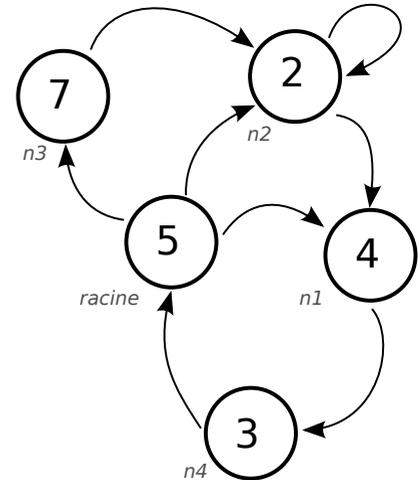


# Mémoire dynamique



Exemple: structure de données avancées: graphe

```
struct noeud
{
    int valeur;
    int nombre_fils;
    struct noeud **fils;
};
```



```
//utilisation
int valeur_racine=racine->valeur;
int valeur_n4=racine->fils[1]->fils[0]->fils[0]->valeur;

printf("%d %d\n",valeur_racine,valeur_n4);
```

# Mémoire dynamique



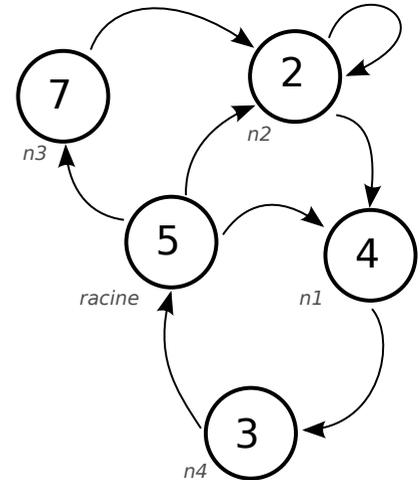
Exemple: structure de données avancées: graphe

```
struct noeud
{
    int valeur;
    int nombre_fils;
    struct noeud **fils;
};
```

```
//deallocation
free(racine->fils);
free(n1->fils);
free(n4->fils);
free(n3->fils);
free(n2->fils);

free(racine);
free(n1);
free(n2);
free(n3);
free(n4);

return 0;
}
```



Ne pas oublier de libérer la mémoire  
=> 10 malloc, donc 10 free !

Vérification par valgrind (absence fuite mémoire)  
Obligatoire!

```
HEAP SUMMARY:
  in use at exit: 0 bytes in 0 blocks
  total heap usage: 10 allocs, 10 frees, 144 bytes allocated

All heap blocks were freed -- no leaks are possible
```

Exercice difficile!

# Chaine de compilation

Compilateur

Warnings

Compilation séparée

Précompilateur

Makefile

Edition de liens et librairies

# Chaine de compilation

```
int main()
{
    int a=3;
    a++;
    int b=a+2;

    char texte[]="j aime l informatique";

    return 0;
}
```



Code (C)

(=langage humain)

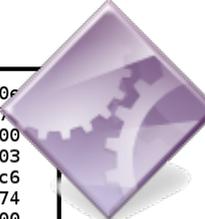
**Compilateur**

Fichier executable

(=binaire)

(Windows: .exe

Linux: nom quelconque)



```
b8af 0860 0055 482d a808 6000 4883 f80e
4889 e577 025d c3b8 0000 0000 4885 c07
f45d bfa8 0860 00ff e00f 1f80 0000 0000
b8a8 0860 0055 482d a808 6000 48c1 f803
4889 e548 89c2 48c1 ea3f 4801 d048 89c6
48d1 fe75 025d c3ba 0000 0000 4885 d274
f45d bfa8 0860 00ff e20f 1f80 0000 0000
803d 4104 2000 0075 1155 4889 e5e8 7eff
ffff 5dc6 052e 0420 0001 f3c3 0f1f 4000
4883 3d08 0220 0000 741b b800 0000 0048
85c0 7411 55bf 9006 6000 4889 e5ff d05d
e97b ffff ffe9 76ff ffff 9090 5548 89e5
c745 fc03 0000 0083 45fc 018b 45fc 83c0
0289 45f8 c745 e06a 2061 69c7 45e4 6d65
206c c745 e820 696e 66c7 45ec 6f72 6d61
c745 f074 6971 7566 c745 f465 00b8 0000
0000 5dc3 9090 9090 9090 9090 9090 9090
4889 6c24 d84c 8964 24e0 488d 2d77 0120
004c 8d25 6801 2000 4889 5c24 d04c 896c
24e8 4c89 7424 f04c 897c 24f8 4883 ec38
4c29 e541 89ff 4989 f648 c1fd 0349 89d5
31db e829 feff ff48 85ed 741a 0f1f 4000
4c89 ea4c 89f6 4489 ff41 ff14 dc48 83c3
0148 39eb 75ea 488b 5c24 0848 8b6c 2410
4c8b 6424 184c 8b6c 2420 4c8b 7424 284c
```

# Compilateur

**gcc**



GNU Compiler Collection

<http://gcc.gnu.org/>

(C, C++, Obj-C, Fortran, Ada, Go)

*Le + répandu sous Linux*

D'autres existent:

*Nwcc*

*Clang*

*Quick C (Microsoft)*

*Turbo C (Embarcadero)*

*XL C (IBM)*

Note:

Compilateur C++ compatibles

*BorlandC++*

*Intel C++ Compiler*

*Visual Studio*

*Turbo C++*

*ProDev*

*Solaris Studio*

# Compilateur

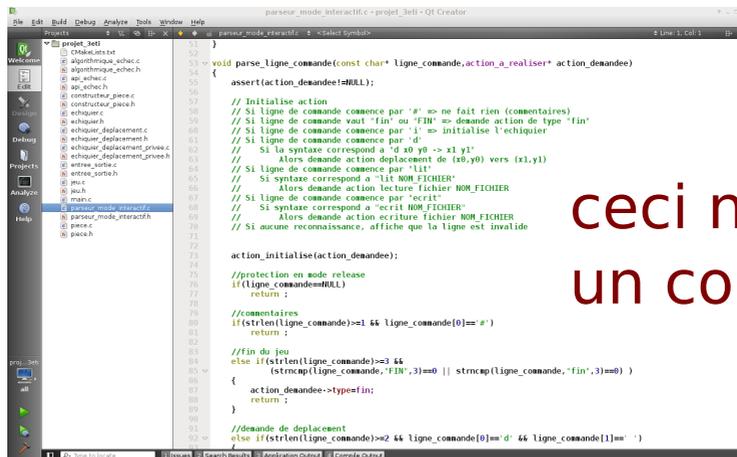
Remarque:

Ne pas confondre **IDE** et **Compilateur**

éditer du texte/code

génère le binaire

Emacs  
QtCreator  
MS Visual C++  
Dev C++  
Eclipse  
Code::Blocks  
Anjuta  
KDevelop



```
void parse_ligne_comande(const char* ligne_comande, action_a_realiser* action_demandee)
{
    assert(action_demandee!=NULL);

    // Initialise action
    // Si ligne de commande commence par '#' => ne fait rien (commentaires)
    // Si ligne de commande veut 'fin' ou 'FIN' => demande action de type 'fin'
    // Si ligne de commande commence par '.' => initialise l'echiquier
    // Si ligne de commande commence par 'd'
    // Si la syntaxe correspond a 'd x0 y0 => x1 y1'
    // Alors demande action deplacement de (x0,y0) vers (x1,y1)
    // Si ligne de commande commence par 'l'
    // Si syntaxe correspond a 'lit NON_FICHER'
    // Alors demande action lecture fichier NON_FICHER
    // Si ligne de commande commence par 'e'
    // Si syntaxe correspond a 'ecrit NON_FICHER'
    // Alors demande action ecriture fichier NON_FICHER
    // Si aucune reconnaissance, affiche que la ligne est invalide

    action_initialise(action_demandee);

    //protection en mode release
    if(ligne_comande==NULL)
        return;

    //commentaires
    if(strlen(ligne_comande)==1 && ligne_comande[0]=='#')
        return;

    //fin du jeu
    else if(strlen(ligne_comande)==3 &&
            (strcmp(ligne_comande,'FIN')==0 || strcmp(ligne_comande,'fin')==0))
    {
        action_demandee->type=fin;
        return;
    }

    //demande de deplacement
    else if(strlen(ligne_comande)==2 && ligne_comande[0]=='d' && ligne_comande[1]==' ')
```

ceci n'est pas  
un compilateur!



# Compilateur: Paramètres de GCC

gcc possède des paramètres:

- Warnings
- Debug
- Optimisation
  
- Chemins d'accès
- Lien avec bibliothèques
- Constantes
- ...

beaucoup d'options!

<http://gcc.gnu.org/onlinedocs/gcc/Option-Summary.html>

# Chaine de compilation

Compilateur

→ **Warnings**

Compilation séparée

Précompilateur

Makefile

Edition de liens et librairies

# Compilateur

## Ex: Warning

```
#include <stdio.h>

int main()
{
    float *a;
    printf("%f", *a);

    return 0;
}
```

sans option de Warning

```
[damien@garonne ~/work/2012_2013_teaching/2012_3eti_projet_c/cours/src/code]$ gcc compilation.c
[damien@garonne ~/work/2012_2013_teaching/2012_3eti_projet_c/cours/src/code]$ gcc compilation.c -Wuninitialized
compilation.c: In function 'main':
compilation.c:7:17: warning: 'a' is used uninitialized in this function [-Wuninitialized]
```

avec option de Warning

en francais:

compilation.c, ligne 17. Warning 'a' est utilisé sans être initialisé dans cette fonction.

```
Warning = Aide pour le programmeur
         = Lisible pour le programmeur
         = Evite les erreurs "silencieuses"
```

# Warnings typiques:

```
int oublie_retour(int valeur)
{
    int valeur_a_retourner=0;
    valeur_a_retourner=valeur+1;
}

int* recupere_adresse_temporaire(int valeur)
{
    return &valeur;
}

int main()
{
    int a=oublie_retour(3);
    printf("%d\n",a);

    int tableau[2]={1,2,3};

    int b=3;
    int *p=recupere_adresse_temporaire(b);
    printf("%d\n",*p);

    int somme=0;
    unsigned int k=0;
    for(k=5;k>0;--k)
        somme += 3;

    return 0;
}
```

Sans warnings

```
$ gcc mon_fichier.c
```

compile

Fonctionne?

# Warnings typiques:

```
int oublie_retour(int valeur)
{
    int valeur_a_retourner=0;
    valeur_a_retourner=valeur+1;
}

int* recupere_adresse_temporaire(int valeur)
{
    return &valeur;
}

int main()
{
    int a=oublie_retour(3);
    printf("%d\n",a);

    int tableau[2]={1,2,3};

    int b=3;
    int *p=recupere_adresse_temporaire(b);
    printf("%d\n",*p);

    int somme=0;
    unsigned int k=0;
    for(k=5;k>0;--k)
        somme += 3;

    return 0;
}
```

pas de retour

adresse temporaire

depassement tableau

toujours vrai: boucle infinie

Ajout de warnings

```
$ gcc mon_fichier.c -Wall -Wextra
```

```
mon_fichier.c: In function 'oublie_retour':
mon_fichier.c:6:9: warning: variable 'valeur_a_retourner' set but not used [-Wunused-but-set-variable]
mon_fichier.c: In function 'recupere_adresse_temporaire':
mon_fichier.c:12:5: warning: function returns address of local variable [enabled by default]
mon_fichier.c: In function 'main':
mon_fichier.c:20:5: warning: excess elements in array initializer [enabled by default]
mon_fichier.c:20:5: warning: (near initialization for 'tableau') [enabled by default]
mon_fichier.c:20:9: warning: unused variable 'tableau' [-Wunused-variable]
mon_fichier.c:28:5: warning: comparison of unsigned expression >= 0 is always true [-Wtype-limits]
mon_fichier.c: In function 'oublie_retour':
mon_fichier.c:8:1: warning: control reaches end of non-void function [-Wreturn-type]
```

**Toujours** activer un maximum de Warnings

**-Wall -Wextra**

conteneurs de multiples warnings

**Indispensable!!**

=> Toujours compiler avec `$gcc -Wall -Wextra`

D'autres Warnings très utiles:

(non compris dans -Wall ni -Wextra)

`-Wfloat-equal -Wshadow -Wswitch-default -Wswitch-enum  
-Wwrite-strings -Wpointer-arith -Wcast-qual -Wredundant-decls  
-Winit-self`

# Warnings : Bonnes pratiques

**Toujours** activer un maximum de Warnings

Ne **jamais** se priver du travail du compilateur!

Si après analyse des Warnings inutiles gachent la **lisibilité**  
(ex. unused variables)

Option:  -Wno-<nom\_warning>

ex. -Wall -Wextra -Wno-unused-variable

# Warnings : Bonnes pratiques

gcc -Wall -Wextra -Wfloat-equal -Wshadow -Wswitch-default -Wswitch-enum -Wwrite-strings  
-Wpointer-arith -Wcast-qual -Wredundant-decls -Winit-self

## Trop long à écrire ?

→ 1 unique fois dans un fichier  
utilisez les scripts

**script** = lignes de commandes écrites dans un fichier  
exécutées les unes derrière les autres

ex.



<nom\_script>.sh

```
#!/bin/bash  
  
<mes_lignes_de_scripts>  
<...>
```

# Script pour compiler

En pratique:

1. créez le fichier:



mon\_script.sh

dans le même repertoire que:



mon\_programme.c

2. y inscrire:

```
#!/bin/bash  
gcc mon_fichier.c -Wall -Wextra -o mon_executable
```

3. rendez le fichier executable: `$ chmod +x mon_script.sh`

4. lancez le: `$ ./mon_script.sh`

# Script pour compiler

En pratique:

Avec l'ensemble des paramètres de compilation:

```
#!/bin/bash
CFLAGS="-Wall -Wextra -Wfloat-equal -Wshadow -Wswitch-default -Wswitch-enum -Wwrite-strings"
gcc mon_fichier.c -o mon_executable ${CFLAGS}
```

déclaration d'une variable (+ lisible)

utilisation d'une variable

**Note:** En anglais: options de compilation = *flags*

options de compilation du compilateur C = *Cflags*

version générique:

```
#!/bin/bash
CFLAGS="-Wall -Wextra -Wfloat-equal -Wshadow -Wswitch-default -Wswitch-enum -Wwrite-strings"
nom_de_fichier=mon_fichier.c
echo "Je compile" ${nom_de_fichier}
gcc ${nom_de_fichier} -o mon_executable ${CFLAGS}
```

**-g**

**Toujours** compiler avec l'option de Debug lors du développement

Permet l'utilisation des debuggers (gdb,kdbg,ddd,valgrind, ...)

Pour résumer, CFLAGS minimales à toujours utiliser:

**-Wall -Wextra -g**

# Chaine de compilation

Compilateur

Warnings

→ **Compilation séparée**

Précompilateur

Makefile

Edition de liens et librairies

# Compilation séparée

1 Fichier:



fichier.c

`gcc -Wall -Wextra -g  
fichier.c -o executable`



executable

2 Fichiers:



f1.c



f2.c

?

?



executable

exec1

exec2

# Cas de 2 fichiers

Un executable => 1 point d'entrée (int main())

## Cas 1:



f1.c

```
int main()
{
    printf("Je suis f1.c");
    return 0;
}
```



f2.c

```
int main()
{
    printf("Je suis f2.c");
    return 0;
}
```

```
$ gcc -Wall -Wextra -g f1.c -o executable_1
$ gcc -Wall -Wextra -g f2.c -o executable_2
```

2 créations  
d'executables  
distincts



executable\_1

```
$ ./executable_1
> Je suis f1.c
```



executable\_2

```
$ ./executable_2
> Je suis f2.c
```

# Cas de 2 fichiers

## Cas 2:

f1.c

```
#include "f2.h"

int main()
{
    printf("Je suis f1.c");
    fonction_de_f2();

    return 0;
}
```

f2.h

```
void fonction_de_f2();
```

f2.c

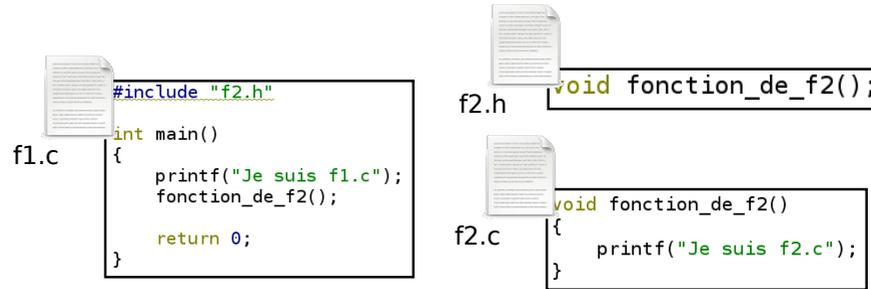
```
void fonction_de_f2()
{
    printf("Je suis f2.c");
}
```

f2 ne contient pas de fonction: *int main()*  
=> pas d'executable associé à f2 uniquement!

f2 agit en tant que "*librairie*" pour le code appelé dans f1

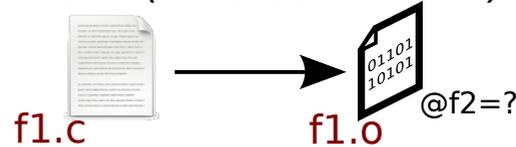
# Cas de 2 fichiers: compilation séparée

## Cas 2:

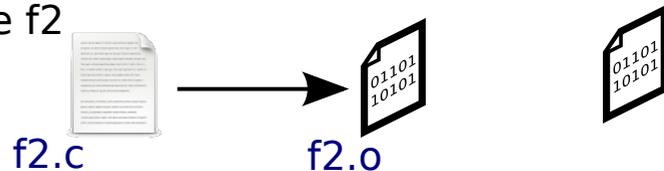


Principe:

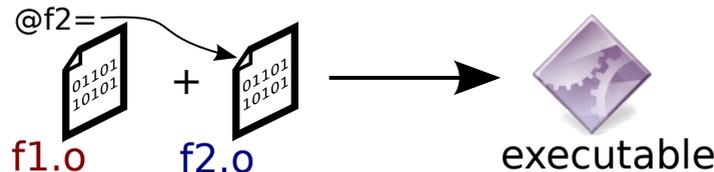
- On compile tout ce qu'on peut de f1  
=> *cad: tout sauf la fonction de f2 (adresse à définir)*



- On compile tout ce qu'on peut de f2  
=> *cad: tout*



- On rassemble les 2 binaires + complète l'adresse de fonction définitive

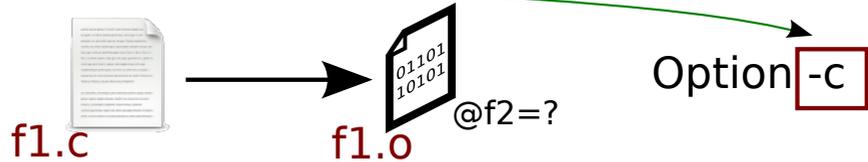


# Cas de 2 fichiers: compilation séparée

## Cas 2, ligne de commande:

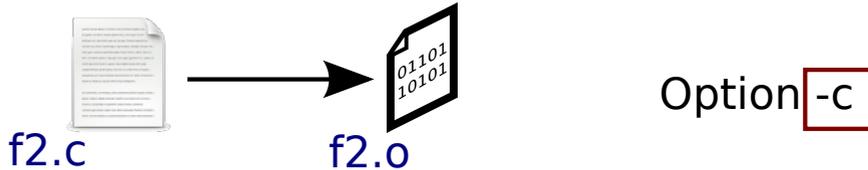
```
gcc -c -Wall -Wextra -g f1.c -o f1.o
```

- Compilation de f1.c



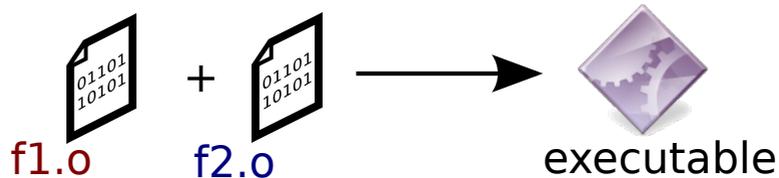
```
gcc -c -Wall -Wextra -g f2.c -o f2.o
```

- Compilation de f2.c



- Edition de liens, génération d'executable

```
gcc f1.o f2.o -o executable
```



# Fichier objet



```
gcc -c -Wall -Wextra -g f1.c -o f1.o
```

Option `-c` indique:  
génère un binaire du code  
ne génère pas d'exécutable  
s'arrête avant l'étape d'édition de lien



fichier objet = fichier binaire

fichier non exécutable

peut contenir des liens vers des fonctions externes

```
7f45 4c46 0201 0100 0000 0000 0000 0000
0100 3e00 0100 0000 0000 0000 0000 0000
0000 0000 0000 0000 4001 0000 0000 0000
0000 0000 4000 0000 0000 4000 0000 0a00
5548 89e5 bf00 0000 00b8 0000 0000 e800
0000 00b8 0000 0000 e800 0000 00b8 0000
0000 5dc3 4a65 2073 7569 7320 6631 2e63
0000 4743 433a 2028 474e 5529 2034 2e37
2e31 2032 3031 3230 3732 3120 2870 7265
7265 6c65 6173 6529 0000 0000 0000 0000
1400 0000 0000 0000 017a 5200 0178 1001
1b0c 0708 9001 0000 1c00 0000 1c00 0000
0000 0000 2400 0000 0041 0e10 8602 430d
065f 0c07 0800 0000 002e 7379 6d74 6162
002e 7374 7274 6162 002e 7368 7374 7274
6162 002e 7265 6c61 2e74 6578 7400 2e64
6174 6100 2e62 7373 002e 726f 6461 7461
002e 636f 6d6d 656e 7400 2e6e 6f74 652e
474e 552d 7374 6163 6b00 2e72 656c 612e
6568 5f66 7261 6d65 0000 0000 0000 0000
```

# Fichier objet

Pour visualiser un fichier binaire

ex.

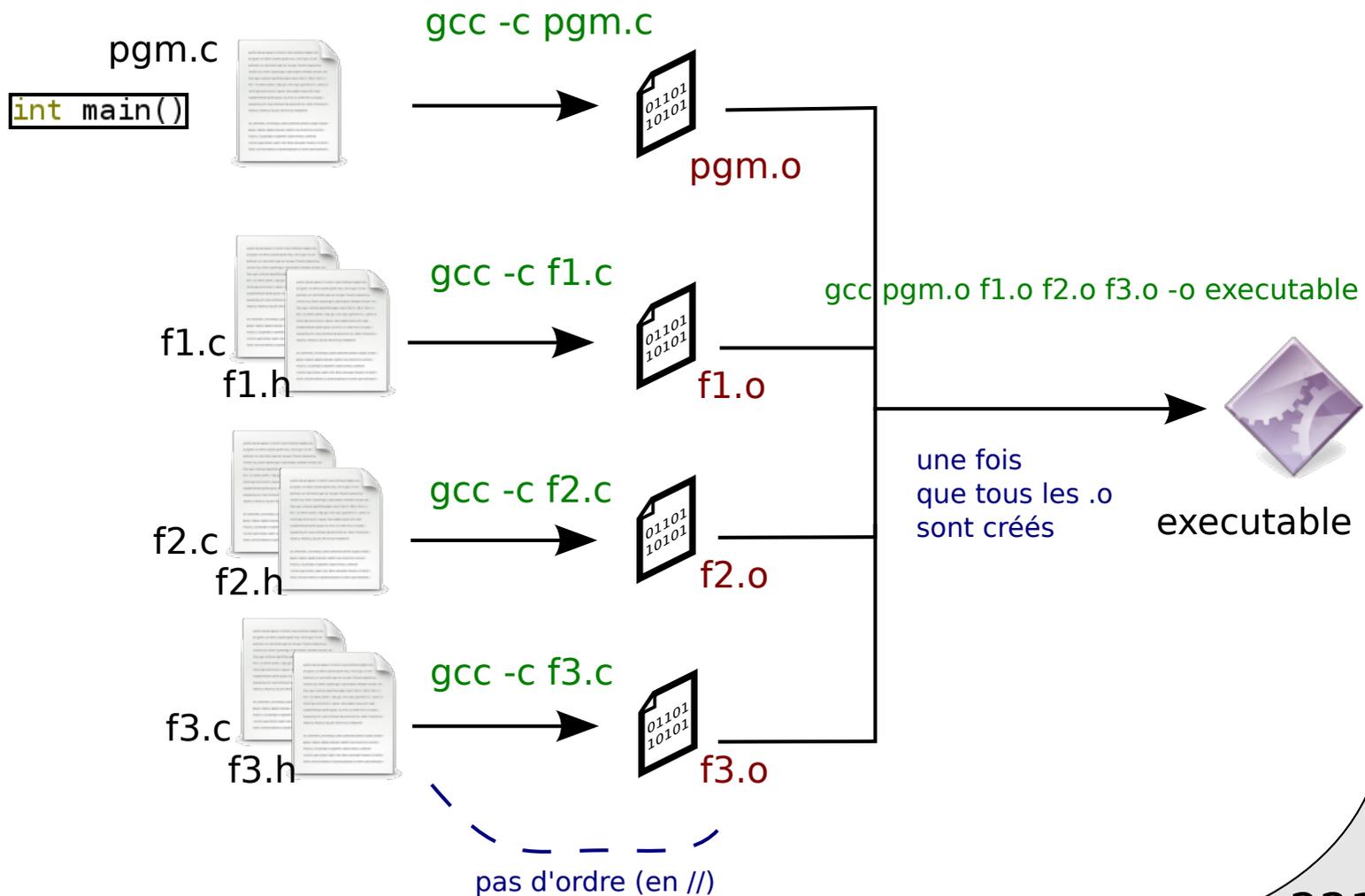
Ouvrir le fichier avec xemacs ou emacs  
`$ xemacs fichier.o`

Appuyez sur **Alt+X**: cherchez *hexl-mode*

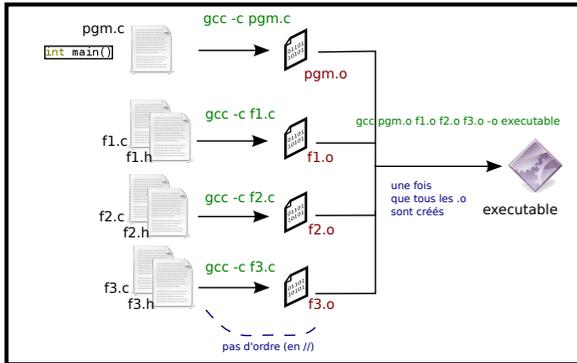
ou

`$ od -x fichier.o`

# Plusieurs sources, un exécutable



# Plusieurs sources, un exécutable: script



mon\_script\_de\_compilation.sh

```
#!/bin/bash
```

```
CFLAGS="-Wall -Wextra -Wfloat-equal -Wshadow -Wswitch-default -Wswitch-enum -Wwrite-strings -Wpointer-arith -Wcast-qual -Wredundant-decls -Winit-self -g"
```

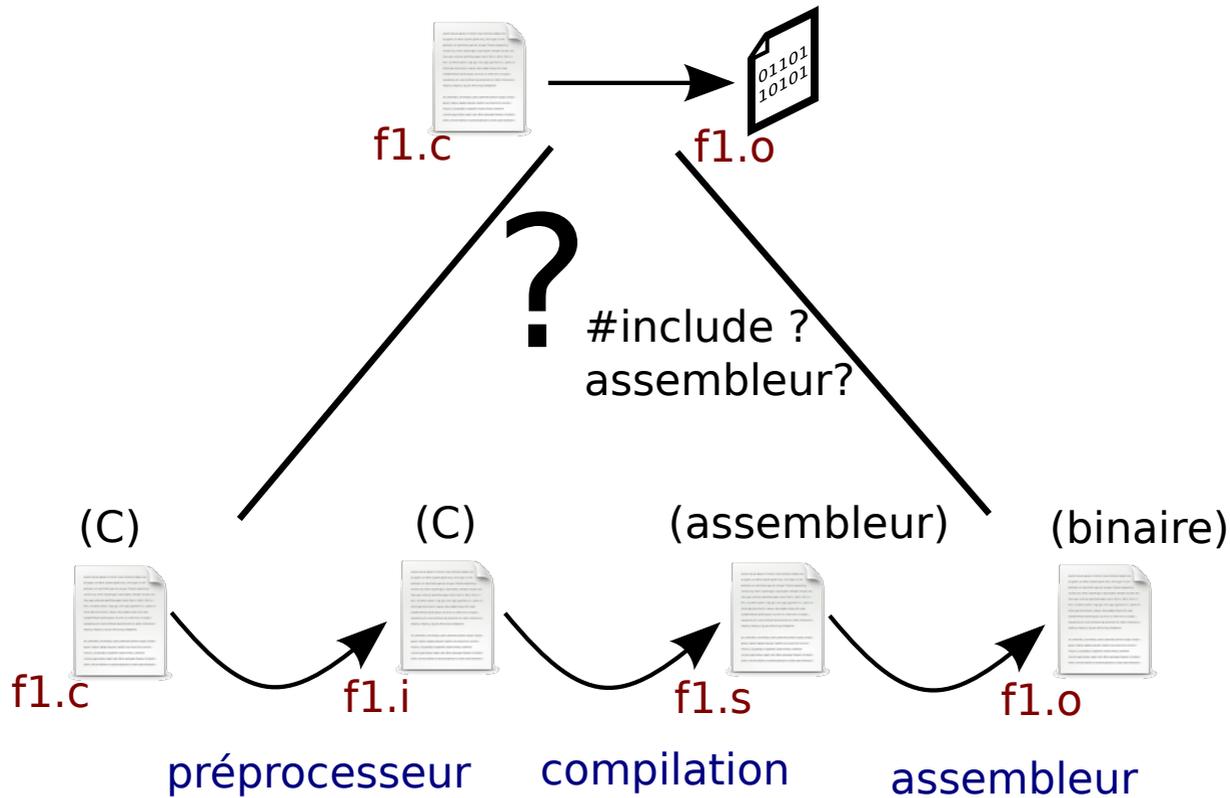
```
#Compilation vers fichiers objets
```

```
gcc f1.c -o f1.o ${CFLAGS}  
gcc f2.c -o f2.o ${CFLAGS}  
gcc f3.c -o f3.o ${CFLAGS}  
gcc pgm.c -o pgm.o ${CFLAGS}
```

```
#Edition de liens
```

```
gcc f1.o f2.o f3.o pgm.o -o executable
```

# Construction d'un fichier objet



# Fichier assembleur



## Option **-S**

```
int main()
{
    int a=5;
    int b=6;
    int c=a+b;
}
```

```
.file "compilation.c"
.text
.globl main
.type main, @function

main:
.LFB0:
.cfi_startproc
pushq %rbp
.cfi_def_cfa_offset 16
.cfi_offset 6, -16
movq %rsp, %rbp
.cfi_def_cfa_register 6
movl $5, -4(%rbp)
movl $6, -8(%rbp)
movl -8(%rbp), %eax
movl -4(%rbp), %edx
addl %edx, %eax
movl %eax, -12(%rbp)
popq %rbp
.cfi_def_cfa 7, 8
ret
.cfi_endproc

.LFE0:
.size main, .-main
.ident "GCC: (GNU) 4.7.1 20120721 (prerelease)"
.section .note.GNU-stack,"",@progbits
```

# Chaine de compilation

Compilateur

Warnings

Compilation séparée

→ **Précompilateur**

Makefile

Edition de liens et librairies

# Precompilateur

```
gcc -E f1.c > f1.i
```

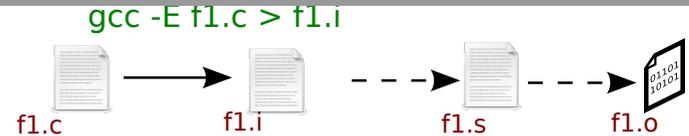


Option **-E**

Precompilateur =

Conversion du code C en un autre code C (plus simple à compiler)

# Precompilateur



## Elimine tous les commentaires

=> inutile pour générer de l'assembleur

f1.c

```
/**
 * Ceci est un commentaire qui devrait
 * décrire mon programme.
 * J'aime l'informatique.
 * J'aime le C.
 */
int main()
{
    // ceci est un commentaire
    int a=5;
    /** Ceci est un autre commentaire */
    int b=6;
    int c=a+b;
}
```

gcc -E f1.c > f1.i

f1.i

```
# 1 "compilation.c"
# 1 "<command-line>"
# 1 "compilation.c"

int main()
{
    int a=5;
    int b=6;
    int c=a+b;
}
```

Conclusion:

Ecrivez vos commentaires pour les humains, pas pour la machine!

# Precompilateur

gcc -E f1.c > f1.i



Remplace toute les *Macros*

#define



f1.c

```
#define MA_CONSTANTE 8
#define PI 3.14159

int main()
{
    int a=MA_CONSTANTE;

    int b=MA_CONSTANTE+8;

    int tableau[MA_CONSTANTE];

    float rayon=3.0;
    float aire=2*PI*RAYON;
}
```

gcc -E f1.c > f1.i

f1.i

```
# 1 "compilation.c"
# 1 "<command-line>"
# 1 "compilation.c"

int main()
{
    int a=8;

    int b=8 +8;

    int tableau[8];

    float rayon=3.0;
    float aire=2*3.14159*RAYON;
}
```

Conclusion:

Centralisez vos constantes (tailles tableaux) dans des macros.

# Precompilateur



Remplace toute les *Macros*

`#define`

**Attention:** Les macros sont directement copiés collés!  
=> Pas de points-virgules (;) !



f1.c

```
#define MA_CONSTANTE 8;
#define PI 3.14159;

int main()
{
    int a=MA_CONSTANTE;

    int b=MA_CONSTANTE+8;

    int tableau[MA_CONSTANTE];

    float rayon=3.0;
    float aire=2*PI*RAYON;
}
```

gcc -E f1.c > f1.i

f1.i

```
# 1 "compilation.c"
# 1 "<command-line>"
# 1 "compilation.c"

int main()
{
    int a=8;;

    int b=8;+8;

    int tableau[8;];

    float rayon=3.0;
    float aire=2*3.14159; *RAYON;
}
```

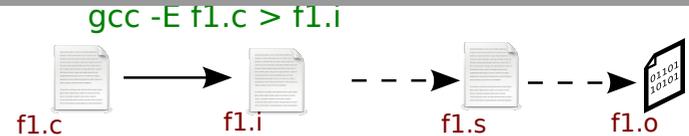
Conclusion:

Pas de points-virgules dans les macros!

# Precompilateur

Remplace toute les *Macros*  
`#define`

Les macros peuvent être des fonctions!



f1.c

```
#define CARRE(x) x*x
#define NORME(x,y) sqrt(CARRE(x)+CARRE(y))

int main()
{
    float a=5;
    float b=CARRE(4);

    float d=NORME(a,b);
    float e=NORME(8,1);
}
```

gcc -E f1.c > f1.i

f1.i

```
# 3 "compilation.c" 2

int main()
{
    float a=5;
    float b=4*4;

    float d=sqrt(a*a+b*b);
    float e=sqrt(8*8 +1*1);
}
```

# Precompilateur



Remplace toute les *Macros*

#define

Les macros peuvent être des fonctions!

Attention au principe du copié-collé des macros !!!

f1.c

```
#define CARRE(x) x*x
#define NORME(x,y) sqrt(CARRE(x)+CARRE(y))

int main()
{
    float a=5;
    float b=CARRE(4+5);

    float d=NORME(a+b,b);
    float e=NORME(8+1,1-4);
}
```

gcc -E f1.c > f1.i

f1.i

```
int main()
{
    float a=5;
    float b=4+5*4+5;

    float d=sqrt(a+b*a+b*b);
    float e=sqrt(8+1*8+1 +1-4*1-4);
}
```

# Precompilateur



Remplace toute les *Macros*  
#define

Les macros peuvent être des fonctions!

Attention au principe du copié-collé des macros !!!

f1.c

```
#define CARRE(x) ((x)*(x))
#define NORME(x,y) (sqrt(CARRE(x)+CARRE(y)))

int main()
{
    float a=5;
    float b=CARRE(4+5);

    float d=NORME(a+b,b);
    float e=NORME(8+1,1-4);
}
```



gcc -E f1.c > f1.i

f1.i

```
int main()
{
    float a=5;
    float b=((4+5)*(4+5));

    float d=(sqrt(((a+b)*(a+b))+((b)*(b))));
    float e=(sqrt(((8+1)*(8+1))+((1-4)*(1-4))));
}
```



=> Autour de chaque variable/expression: ajoutez des parenthèses

# Precompilateur



gcc -E f1.c > f1.i



Remplace toute les *Macros*

#define

Les macros peuvent être des fonctions avancées !

f1.c

```
#define LOOP(k,N) for((k)=0;(k)<(N);++(k))
#define MAKE_VECTOR(vec,x,y,z) float vec[3];\
    vec[0]=(x);vec[1]=(y);vec[2]=(z);
#define PRINT_SUM(vec) {printf("%f\n", (vec[0])+(vec[1])+(vec[2]));}

int main()
{
    int k=0;
    LOOP(k,15)
    {
        MAKE_VECTOR(mon_vecteur,k,2*k,k)
        PRINT_SUM(mon_vecteur)
    }
}
```

gcc -E f1.c > f1.i

f1.i

```
int main()
{
    int k=0;
    for((k)=0;(k)<(15);++(k))
    {
        float mon_vecteur[3]; mon_vecteur[0]=(k);mon_vecteur[1]=(2*k);mon_vecteur[2]=(k);
        {printf("%f\n", (mon_vecteur[0])+(mon_vecteur[1])+(mon_vecteur[2]));}
    }
}
```

# Precompilateur



gcc -E f1.c > f1.i



Nouveau langage?

f1.c

```
DEBUT_PROGRAMME

Initialise(Valeur_utilisateur)
Variable A Recoit Valeur_utilisateur;

SI(A Plus_grand_que 10)
  Affiche("Trop grand");
  Quitte
FIN_SI

Variable K Recoit 0;
TANT_QUE(K Plus_petit_que A)
  Affiche(K)
  Incrmente(K);
FIN_TANT_QUE

FIN_PROGRAMME
```

+

```
#define DEBUT_PROGRAMME int main(){
#define FIN_PROGRAMME return 0;}
#define Recoit =
#define Variable int
#define SI(x) if(x){
#define Plus_grand_que >
#define Plus_petit_que <
#define Affiche(x) {printf("%d\n", (x));}
#define Incrmente(x) x=x+1
#define TANT_QUE(x) while(x){
#define FIN_SI }
#define FIN_TANT_QUE }
#define Initialise(x) int x;\
scanf("%d",&x);
#define Quitte abort();
```

f1.i

```
int main(){

int Valeur_utilisateur; scanf("%d",&Valeur_utilisateur);
int A = Valeur_utilisateur;

if(A > 10){
{printf("%d\n", ("Trop grand"));};
abort();
}

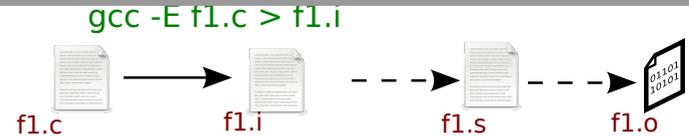
int K = 0;
while(K < A){
{printf("%d\n", (K));}
K=K+1;
}

return 0;}
```

gcc -E f1.c > f1.i

# Precompilateur

Synthèse Macros:



**Avantage:** Pas de types

**Inconvénients:** Difficile à écrire  
Difficile à debugger  
Cas non prévues

**Macro = Attention!**

Ne pas abuser des macros

Règles: Ne pas utiliser une macro si une fonction peut le faire!

Ne pas utiliser de macro si il y a ambiguïté

Ne pas utiliser de macro si le code est moins lisible

# Precompilateur

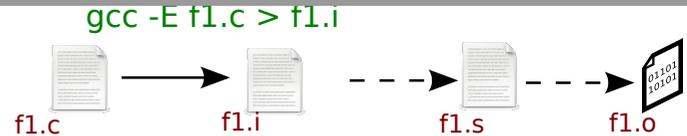
gcc -E f1.c > f1.i



Quelques macros utiles

```
__LINE__,  
__FUNCTION__,  
__FILE__,  
__DATE__
```

# Precompilateur



Quelques macros utiles: exemple d'utilisation

f1.c

```
void affiche_macro_utiles()
{
    printf("%d\n", __LINE__);
    printf("%s\n", __FUNCTION__);
    printf("%s\n", __FILE__);
    printf("%s\n", __DATE__);
    printf("%s\n", __TIME__);
}

int main()
{
    affiche_macro_utiles();
}
```

f1.i

```
void affiche_macro_utiles()
{
    printf("%d\n", 8);
    printf("%s\n", __FUNCTION__);
    printf("%s\n", "compilation.c");
    printf("%s\n", "Sep 7 2012");
    printf("%s\n", "17:59:04");
}

int main()
{
    affiche_macro_utiles();
}
```

executable

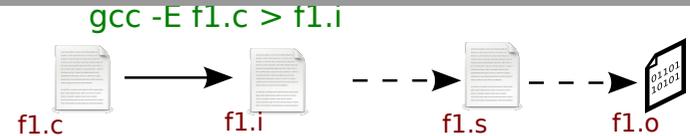
```
8
affiche_macro_utiles
compilation.c
Sep 7 2012
17:57:24
```

gcc -E f1.c > f1.i

gcc -g -Wall -Wextra f1.c -o executable

Rem. `__FUNCTION__` n'est pas une vraie macro, mais une variable liée à gcc

# Precompilateur



Quelques macros utiles: exemple d'utilisation

Utile pour messages d'erreurs!

```
void ma_fonction(int T[])
{
    //boucle complexe
    int k=0
    for(k=0;...)
    {
        ...
        //detection d'erreur
        if(erreur)
        {
            printf("Erreur detectee ligne %d, fonction %s, fichier %s\n",
                __LINE__, __FUNCTION__, __FILE__);
        }
        ...
    }
}
```

# Precompilateur



Cas d'exemple complet: debug avancé



```
#define DEBUG(message,indice) printf("Erreur (%s%d): ligne %d, fonction [%s], fichier [%s]\n",\
    message,indice,__LINE__,__FUNCTION__,__FILE__)\n\nfloat epsilon=1e-5;\n\nstruct v3\n{\n    float x,y,z;\n};\n\nvoid v3_init(struct v3* vec);\nfloat v3_norme(const struct v3* vec);\nvoid v3_divise(struct v3* vec,float denominateur);\nvoid v3_normalise(struct v3* vec);
```

# Precompilateur

gcc -E f1.c > f1.i



## Cas d'exemple complet: debug avancé



```
#define DEBUG(message,indice) printf("Erreur (%s%d): ligne %, fonction [%s], fichier [%s]\n",\
    message,indice, __LINE__, __FUNCTION__, __FILE__)
```

```
void v3_init(struct v3* vec)
{
    vec->x=0; vec->y=0; vec->z=0;
}

float v3_norme(const struct v3* vec)
{
    return sqrt(vec->x*vec->x+vec->y*vec->y+vec->z*vec->z);
}

void v3_divise(struct v3* vec, float denominateur)
{
    vec->x/=denominateur;
    vec->y/=denominateur;
    vec->z/=denominateur;
}

void v3_normalise(struct v3* vec)
{
    v3_divise(vec, v3_norme(vec));
}
```

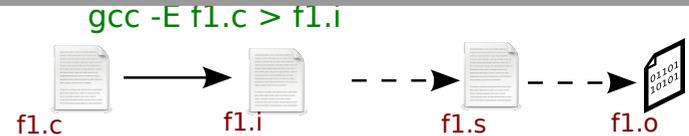
```
int main()
{
    struct v3 vec[5];
    int k=0;
    for(k=0;k<5;++k)
        v3_init(&vec[k]);

    for(k=0;k<4;++k) //oublie d'un vecteur
        vec[k].x=k+1;

    normalise_tableau_de_vecteur(vec);
}
```

```
void normalise_tableau_de_vecteur(struct v3 vec[])
{
    int k=0;
    for(k=0;k<5;++k)
    {
        //normalisation protegee de chaque vecteur
        float norme=v3_norme(&vec[k]);
        if(norme<epsilon)
            DEBUG("norme nulle pour k=",k);
        else
            v3_normalise(&vec[k]);
    }
}
```

# Precompilateur



Cas d'exemple complet: debug avancé



```
#define DEBUG(message,indice) printf("Erreur (%s%d): ligne %, fonction [%s], fichier [%s]\n",\
    message,indice, __LINE__, __FUNCTION__, __FILE__)
```

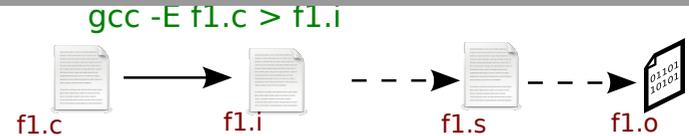
```
void normalise_tableau_de_vecteur(struct v3 vec[])
{
    int k=0;
    for(k=0;k<5;++k)
    {
        //normalisation protegee de chaque vecteur
        float norme=v3_norme(&vec[k]);
        if(norme<epsilon)
            DEBUG("norme nulle pour k=",k);
        else
            v3_normalise(&vec[k]);
    }
}
```

Debug simple  
on sait d'où viens l'erreur

Affichage:

```
Erreur (norme nulle pour k=4): ligne 27,
fonction [normalise_tableau_de_vecteur], fichier [compilation.c]
```

# Precompilateur



## Instructions conditionnelles du precompilateur

```
#if <condition>  
<...>  
#endif
```

//si condition est vraie

```
#if <condition>  
<...>  
#else  
<...>  
#endif
```

//si condition est vraie  
//sinon

```
#if <condition>  
<...>  
#elif <condition 2>  
<...>  
#else  
<...>  
#endif
```

//si condition est vraie  
//sinon, si condition 2 est vraie  
//sinon

```
#ifdef <variable>  
<...>  
#endif
```

//si variable definie

```
#ifndef <variable>  
<...>  
#endif
```

//si variable non definie



# Precompilateur



gcc -E f1.c > f1.i



Exemple:

f1.c

```
#define A 3

#if A==3
#define B 5
#else
#define B 8
#endif

int main()
{
    int variable=B;

    printf("%d\n",variable);
}
```

f1.i

```
int main()
{
    int variable=5;

    printf("%d\n",variable);
}
```

gcc -E f1.c > f1.i

# Precompilateur



Exemple:  
f1.c

```
#define AVEC_MATH

#ifdef AVEC_MATH
#include <math.h>
#endif

int main()
{
    float x=1e-5;
#ifdef AVEC_MATH
    float sin_x=x;
#else
    float sin_x=sin(x);
#endif
}
```

gcc -E f1.c > f1.i

f1.i

```
int main()
{
    float x=1e-5;
    float sin_x=sin(x);
}
```

f2.c

```
#ifdef AVEC_MATH
#include <math.h>
#endif

int main()
{
    float x=1e-5;
#ifdef AVEC_MATH
    float sin_x=x;
#else
    float sin_x=sin(x);
#endif
}
```

gcc -E f2.c > f2.i

f2.i

```
int main()
{
    float x=1e-5;
    float sin_x=x;
}
```



# Precompilateur

gcc -E f1.c > f1.i



Exemple:

```
#define DEBUG_MODE

#ifndef DEBUG_MODE
#define PRINT_DEBUG printf("Erreur: l.%d, fct:<%s>, fichier:[%s] \n",__LINE__,__FUNCTION__,__FILE__)
#else
#define PRINT_DEBUG
#endif

int divide(int a,int b)
{
    if(b!=0)
        return a/b;
    else
    {
        PRINT_DEBUG;
        return -1;
    }
}

int main()
{
    divide(5,0);

    return 0;
}
```

gcc -E f1.c > f1.i

```
int divide(int a,int b)
{
    if(b!=0)
        return a/b;
    else
    {
        printf("Erreur: l.%d, fct:<%s>, fichier:[%s] \n",17,__FUNCTION__,"compilation.c");
        return -1;
    }
}

int main()
{
    divide(5,0);

    return 0;
}
```

# Precompilateur

gcc -E f1.c > f1.i



Utilisation des conditionnelles+macro:

Pour la portabilité entre systèmes

Pour le debug



```
#if SYSTEM=LINUX
#include <stdio.h>
#else if SYSTEM==WINDOWS
#include <stdafx.h>
#endif

#if ARCHI==x86
...
#else if ARCHI==itanium
...
#else if ARCHI==power_pc
...
#endif

#if NBITS==64
...
#else if NBITS==32
...
#endif
```

Attention: Ne **pas** utiliser pour l'optimisation

Réduit la lisibilité du code.

Si faisable avec une fonction: **utiliser une fonction**

# Precompilateur

gcc -E f1.c > f1.i



Macro: `#include "fichier"`

↑ Copie-collé le contenu d'un fichier



```
mon_fichier
void ma_fonction(int a)
{
    printf("bonjour %d\n", a);
}
int ma_variable=36;
```

```
f1.i
# 1 "compilation.c"
# 1 "<command-line>"
# 1 "compilation.c"

# 1 "mon_fichier" 1
void ma_fonction(int a)
{
    printf("bonjour %d\n", a);
}

int ma_variable=36;
# 3 "compilation.c" 2

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

```
f1.c
#include "mon_fichier"

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

gcc -E f1.c > f1.i

execution:

bonjour 36

# Precompilateur

gcc -E f1.c > f1.i



## En-tête standards



f1.c

```
#include <stdio.h>

int main()
{
}
```

f1.i

```
...
extern int fprintf (FILE *__restrict __stream,
  const char *__restrict __format, ...);

extern int printf (const char *__restrict __format, ...);

extern int sprintf (char *__restrict __s,
  const char *__restrict __format, ...) __attribute__ ((__nothrow__));

...
```

gcc -E f1.c > f1.i

# Precompilateur

gcc -E f1.c > f1.i



## En-tête standards

f1.c

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

f1.i

```
...
extern int fprintf (FILE *__restrict __stream,
                  const char *__restrict __format, ...);

extern int printf (const char *__restrict __format, ...);

extern int sprintf (char *__restrict __s,
                  const char *__restrict __format, ...) __attribute__ ((__nothrow__));

...
```



gcc -E f1.c > f1.i

Note: **#include "NOM"** → recherche fichier dans le repertoire locale

**#include <NOM>** → recherche fichier dans les repertoires systemes (/usr/include/ ; /usr/local/include/ ; ...)



Pour inclure d'autres chemins:  
gcc -I<CHEMIN>

# Precompilateur

gcc -E f1.c > f1.i



## En-tête standards

f1.c

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

gcc -E f1.c > f1.i

f1.i

```
...
extern int fprintf (FILE *__restrict __stream,
  const char *__restrict __format, ...);

extern int printf (const char *__restrict __format, ...);
extern int sprintf (char *__restrict __s,
  const char *__restrict __format, ...) __attribute__ ((__nothrow__));

...

```

Note: Les fichiers d'en tête standards sont volumineux

=> N'inclure que ceux qui sont nécessaire

Les variables des fichiers standards commencent par `_` ou `__`

=> Ne pas utiliser cette convention pour vos variables

# Precompilateur

gcc -E f1.c > f1.i



## Inclusion multiples: *include guard*

### voiture.h

```
//voiture de location
struct voiture
{
    int essence;
    int kilometrage;
};

void voiture_init(struct voiture* v);
```

### trajet.h

```
#include "voiture.h"

void trajet_lyon_paris(struct voiture* voiture);
void trajet_marseille_lyon(struct voiture* voiture);
```

### main.c

```
#include "voiture.h"
#include "trajet.h"

int main()
{
    struct voiture voiture_1;
    voiture_init(&voiture_1);
    trajet_lyon_paris(&voiture_1);
}
```

### voiture.c

```
#include "voiture.h"

void voiture_init(struct voiture* v)
{
    v->kilometrage=0;
    v->essence=60;
}
```

### trajet.c

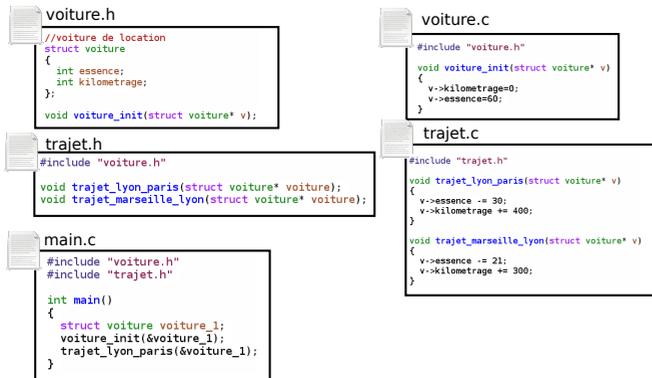
```
#include "trajet.h"

void trajet_lyon_paris(struct voiture* v)
{
    v->essence -= 30;
    v->kilometrage += 400;
}

void trajet_marseille_lyon(struct voiture* v)
{
    v->essence -= 21;
    v->kilometrage += 300;
}
```

# Precompilateur

## Inclusion multiples: *include guard*



gcc -E main.c > main.i

gcc -c main.c -Wall -Wextra -g

```
In file included from trajet.h:5:0,
    from f1.c:3:
voiture.h:6:8: error: redefinition of 'struct voiture'
voiture.h:6:8: note: originally defined here
In file included from trajet.h:5:0,
    from f1.c:3:
voiture.h:12:6: error: conflicting types for 'voiture_init'
In file included from f1.c:2:0:
voiture.h:12:6: note: previous declaration of 'voiture_init' was here
```

gcc -E f1.c > f1.i



main.i

```
# 1 "f1.c"
# 1 "<command-line>"
# 1 "f1.c"
# 1 "voiture.h" 1
struct voiture
{
    int essence;
    int kilometrage;
};
void voiture_init(struct voiture* v);
# 3 "f1.c" 2
# 1 "trajet.h" 1
# 1 "voiture.h" 1
struct voiture
{
    int essence;
    int kilometrage;
};
void voiture_init(struct voiture* v);
# 6 "trajet.h" 2
void trajet_lyon_paris(struct voiture* voiture);
void trajet_marseille_lyon(struct voiture* voiture);
# 4 "f1.c" 2
int main()
{
    struct voiture voiture_1;
    voiture_init(&voiture_1);
    trajet_lyon_paris(&voiture_1);
}
```

inclusion de voiture.h

inclusion de trajet.h

# Precompilateur

## Inclusion multiples: *include guard*

```
voiture.h
//voiture de location
struct voiture
{
    int essence;
    int kilometrage;
};
void voiture_init(struct voiture* v);

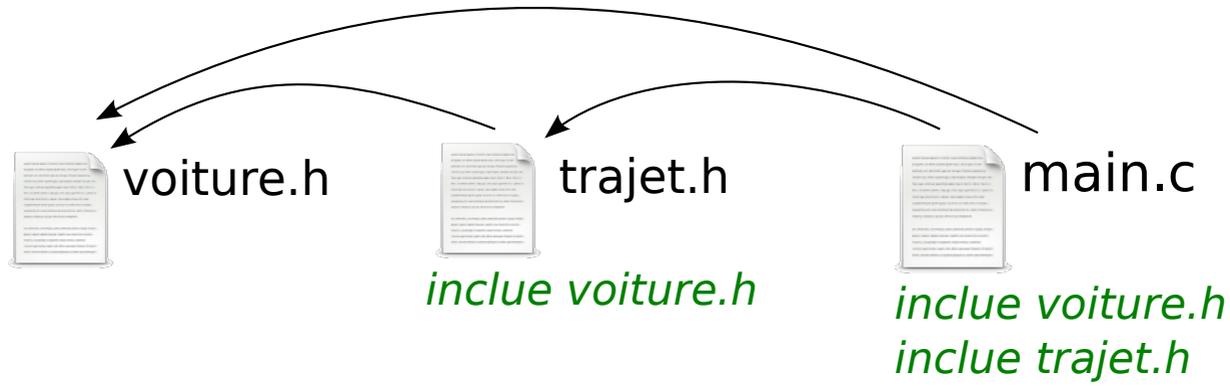
trajet.h
#include "voiture.h"
void trajet_lyon_paris(struct voiture* voiture);
void trajet_marseille_lyon(struct voiture* voiture);

main.c
#include "voiture.h"
#include "trajet.h"
int main()
{
    struct voiture voiture_1;
    voiture_init(&voiture_1);
    trajet_lyon_paris(&voiture_1);
}

voiture.c
#include "voiture.h"
void voiture_init(struct voiture* v)
{
    v->kilometrage=0;
    v->essence=60;
}

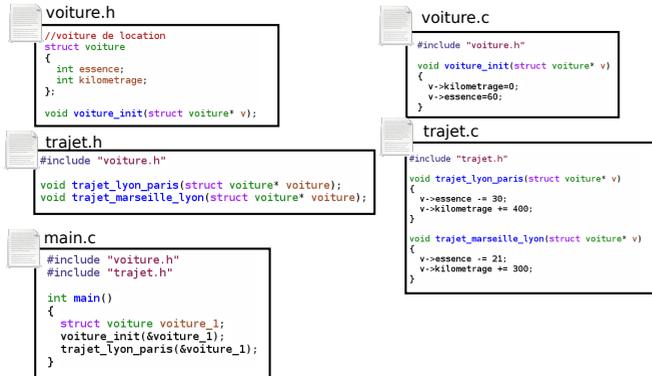
trajet.c
#include "trajet.h"
void trajet_lyon_paris(struct voiture* v)
{
    v->essence -= 30;
    v->kilometrage += 400;
}
void trajet_marseille_lyon(struct voiture* v)
{
    v->essence -= 21;
    v->kilometrage += 300;
}
```

gcc -E f1.c > f1.i



# Precompilateur

## Inclusion multiples: *include guard*

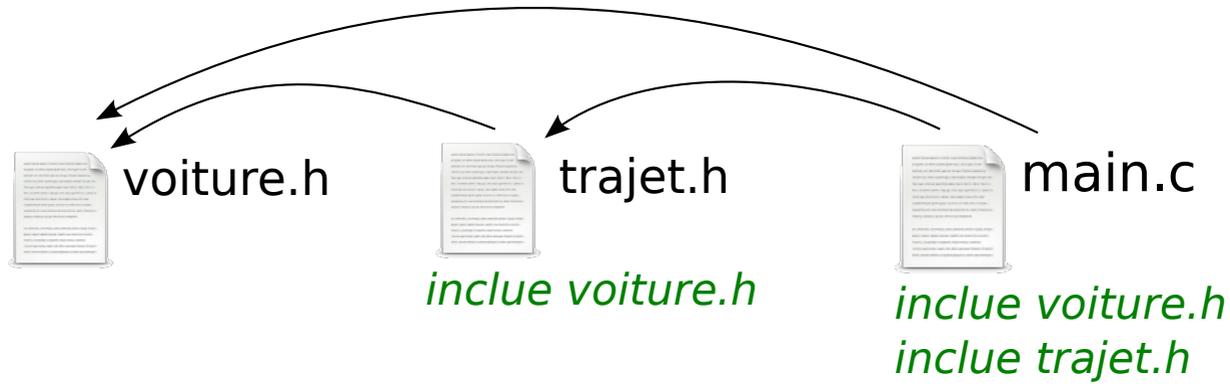


gcc -E f1.c > f1.i



**Solution 1:**  
N'inclure que trajet.h dans main.c  
=> **Ingérable pour un grand projet**

**Solution 2:**  
Compilateur n'inclue pas voiture.h deux fois  
=> **Principe des *includes guards***



# Precompilateur

gcc -E f1.c > f1.i



Inclusion multiples: *include guard*



voiture.h

```
#ifndef VOITURE_H
#define VOITURE_H

//voiture de location
struct voiture
{
    int essence;
    int kilometrage;
};

void voiture_init(struct voiture* v);

#endif
```

une unique inclusion



voiture.c

```
#include "voiture.h"
void voiture_init(struct voiture* v)
{
    v->kilometrage=0;
    v->essence=60;
}
```



trajet.c

```
#include "trajet.h"
void trajet_lyon_paris(struct voiture* v)
{
    v->essence -= 30;
    v->kilometrage += 400;
}
void trajet_marseille_lyon(struct voiture* v)
{
    v->essence -= 21;
    v->kilometrage += 300;
}
```

trajet.h

```
#include "voiture.h"
void trajet_lyon_paris(struct voiture* voiture);
void trajet_marseille_lyon(struct voiture* voiture);
```

main.c

```
#include "voiture.h"
#include "trajet.h"
int main()
{
    struct voiture voiture_1;
    voiture_init(&voiture_1);
    trajet_lyon_paris(&voiture_1);
}
```

# Precompilateur

gcc -E f1.c > f1.i



Inclusion multiples: *include guard*

```
voiture.h
#ifndef VOITURE_H
#define VOITURE_H

//voiture de location
struct voiture
{
    int essence;
    int kilometrage;
};

void voiture_init(struct voiture* v);

#endif
```

```
voiture.c
#include "voiture.h"

void voiture_init(struct voiture* v)
{
    v->kilometrage=0;
    v->essence=0;
}
```

```
trajet.c
#include "trajet.h"

void trajet_lyon_paris(struct voiture* v)
{
    v->essence -= 30;
    v->kilometrage += 400;
}

void trajet_marseille_lyon(struct voiture* v)
{
    v->essence -= 21;
    v->kilometrage += 300;
}
```

```
trajet.h
#include "voiture.h"

void trajet_lyon_paris(struct voiture* voiture);
void trajet_marseille_lyon(struct voiture* voiture);
```

```
main.c
#include "voiture.h"
#include "trajet.h"

int main()
{
    struct voiture voiture_1;
    voiture_init(&voiture_1);
    trajet_lyon_paris(&voiture_1);
}
```

```
main.i
# 1 "f1.c"
# 1 "<command-line>"
# 1 "f1.c"
# 1 "voiture.h" 1
struct voiture
{
    int essence;
    int kilometrage;
};
void voiture_init(struct voiture* v);
# 3 "f1.c" 2
# 1 "trajet.h" 1
void trajet_lyon_paris(struct voiture* voiture);
void trajet_marseille_lyon(struct voiture* voiture);
# 4 "f1.c" 2
int main()
{
    struct voiture voiture_1;
    voiture_init(&voiture_1);
    trajet_lyon_paris(&voiture_1);
}
```

gcc -E main.c > main.i

# Precompilateur

Inclusion multiples: *include guard*



Bonne pratique:

Pour tout fichier d'en-tête de nom: **NOM.h**  
Placez un *include guard*



```
#ifndef NOM_H  
#define NOM_H  
...  
#endif
```

Autres possibilités:

```
#ifndef INCLUDE_GUARD_NOM_H  
#ifndef NOM
```

**But:** Nom éviter les collisions

Attention: Evitez les macros commençant par \_

```
#ifndef _NOM_H
```

# Precompilateur

Exemple réel:  
fichier unistd.h

```
/*
 *      POSIX Standard: 2.10 Symbolic Constants      <unistd.h>
 */

#ifndef _UNISTD_H
#define _UNISTD_H      1

#include <features.h>

__BEGIN_DECLS

/* These may be used to determine what facilities are present at compile time
   Their values can be obtained at run time from `sysconf'.  */

#ifdef __USE_XOPEN2K8
/* POSIX Standard approved as ISO/IEC 9945-1 as of September 2008.  */
# define _POSIX_VERSION 200809L
#elif defined __USE_XOPEN2K
/* POSIX Standard approved as ISO/IEC 9945-1 as of December 2001.  */
# define _POSIX_VERSION 200112L
#elif defined __USE_POSIX199506
/* POSIX Standard approved as ISO/IEC 9945-1 as of June 1995.  */
# define _POSIX_VERSION 199506L
#elif defined __USE_POSIX199309
/* POSIX Standard approved as ISO/IEC 9945-1 as of September 1993.  */
# define _POSIX_VERSION 199309L
#else
/* POSIX Standard approved as ISO/IEC 9945-1 as of September 1990.  */
# define _POSIX_VERSION 199009L
#endif

/* These are not #ifdef __USE_POSIX2 because they are
   in the theoretically application-owned namespace.  */

#ifdef __USE_XOPEN2K8
# define __POSIX2_THIS_VERSION 200809L
/* The utilities on GNU systems also correspond to this version.  */
#elif defined __USE_XOPEN2K
/* The utilities on GNU systems also correspond to this version.  */
# define __POSIX2_THIS_VERSION 200112L
#elif defined __USE_POSIX199506
...

```



## Synthèse précompilateur

Potentiellement puissant (va au delà du langage C)

A utiliser avec précaution

=> rend rapidement le code peu standard  
= peu lisible

### **Bonne pratique:**

### **N'utilisez que les règles standards:**

- \* include guard
- \* #ifdef => portabilité, debug.
- \* pas d'optimisation, préférez les fonctions aux macros

# Chaine de compilation

Compilateur

Warnings

Compilation séparée

Précompilateur

→ **Makefile**

Edition de liens et librairies

# Makefile

**make:** Outil d'automatisation de taches

*Dépendance => Action à réaliser*

**Note:** Makefile est indépendant de gcc!



## Makefile

```
but_1: dependance but 1
    action a realiser pour le but_1

but_2: dependance but 2
    action a realiser pour le but_2
```

# Makefile



## Exemple de Makefile

```
#par default, ce Makefile sert a realiser un fichier du nom de mon_rapport.txt
all: mon_rapport.txt

mon_rapport.txt: partie_1.txt partie_2.txt
    cp partie_1.txt mon_rapport.txt #copie de fichier
    cat partie_2.txt >> mon_rapport.txt #concatenation fichier dans mon_rapport.txt

partie_1.txt:
    echo "Chapitre I: " > partie_1.txt #écriture dans un fichier
    echo "Je realise mon 1er Makefile" >> partie_1.txt

partie_2.txt:
    echo "Chapitre II: " > partie_2.txt
    echo "Je compile mon programme" >> partie_2.txt
```

L'appel à: `$ make`  
génère:



partie\_1.txt



partie\_2.txt



mon\_rapport.txt

# Makefile



## Exemple de Makefile pour compiler du C

variable

```
CFLAGS=-g -Wall -Wextra -Wfloat-equal -Wshadow -Wswitch-default -Wswitch-enum -Wwrite-strings -Wpointer-arith -Wcast-qual -Wredundant-decls -Winit-self

all: mon_executable

mon_executable: f1.o f2.o f3.o
    gcc f1.o f2.o f3.o -o mon_executable

f1.o: f1.c f1.h
    gcc -c f1.c -o f1.o ${CFLAGS}

f2.o: f2.c f2.h
    gcc -c f2.c -o f2.o ${CFLAGS}

f3.o: f3.c f3.h
    gcc -c f3.c -o f3.o ${CFLAGS}
```

Avantage par rapport à un script:

Ne compile que si nécessaire

# Makefile

## Exemple de Makefile pour compiler du C

```
all: mon_executable

mon_executable: f1.o f2.o f3.o
    gcc f1.o f2.o f3.o -o mon_executable

#generation des fichiers assembleurs
assembleur: f1.s f2.s f3.s

f1.o: f1.c f1.h
    gcc -c f1.c -o f1.o ${CFLAGS}

f2.o: f2.c f2.h
    gcc -c f2.c -o f2.o ${CFLAGS}

f3.o: f3.c f3.h
    gcc -c f3.c -o f3.o ${CFLAGS}

#compilation pour assembleur
f1.s: f1.c f1.h
    gcc -c -S f1.c
f2.s: f2.c f2.h
    gcc -c -S f2.c
f3.s: f3.c f3.h
    gcc -c -S f3.c
```

*fichiers assembleurs  
pas de commandes d'executions*

L'appel à  
*\$ make assembleur*

génère les fichiers:  
*f1.s, f2.s, f3.s*

# Chaine de compilation

Compilateur

Warnings

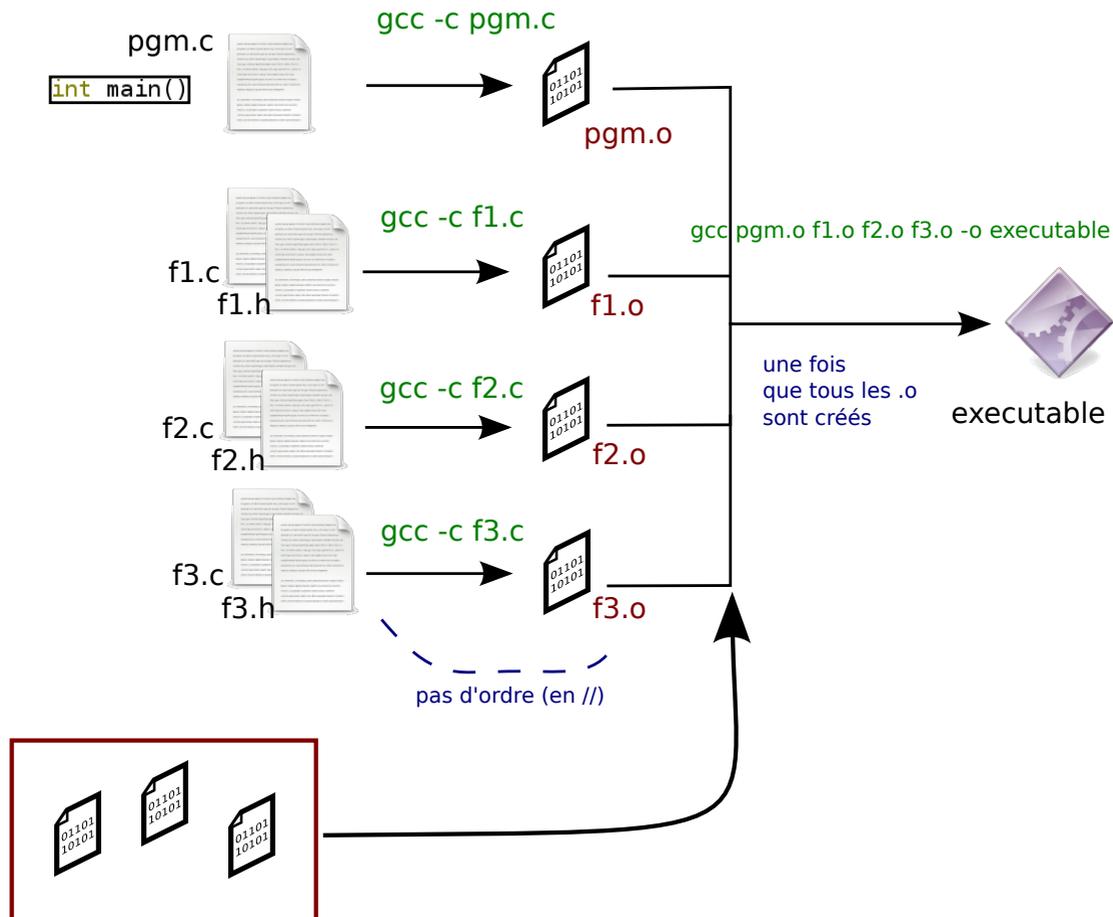
Compilation séparée

Précompilateur

Makefile

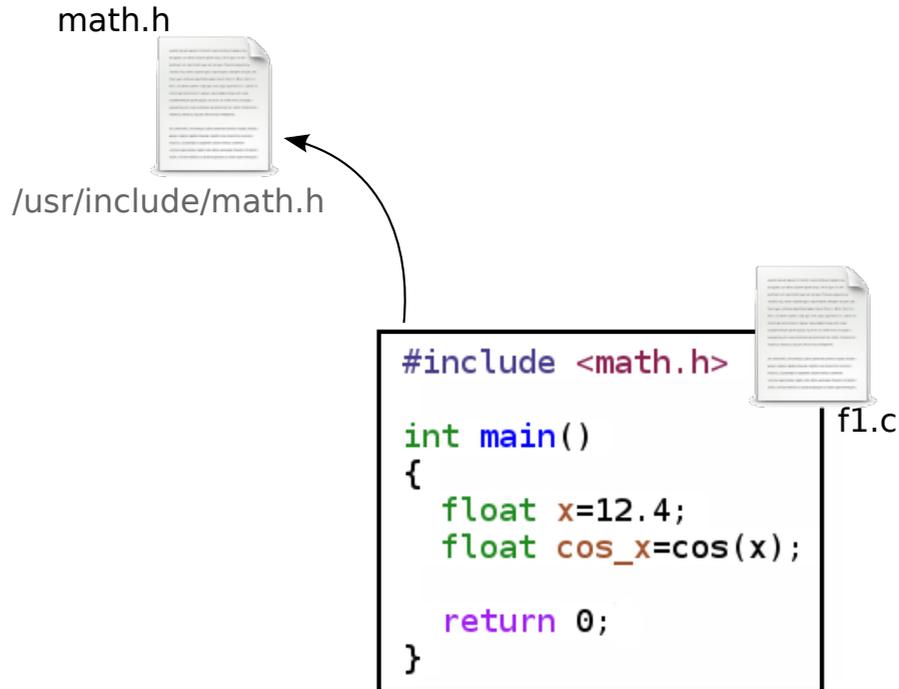
→ **Edition de liens et librairies**

# Edition de liens et bibliothèques



implémentation des bibliothèques externes  
(ex: math, io, posix, ...)

# Edition de liens et bibliothèques



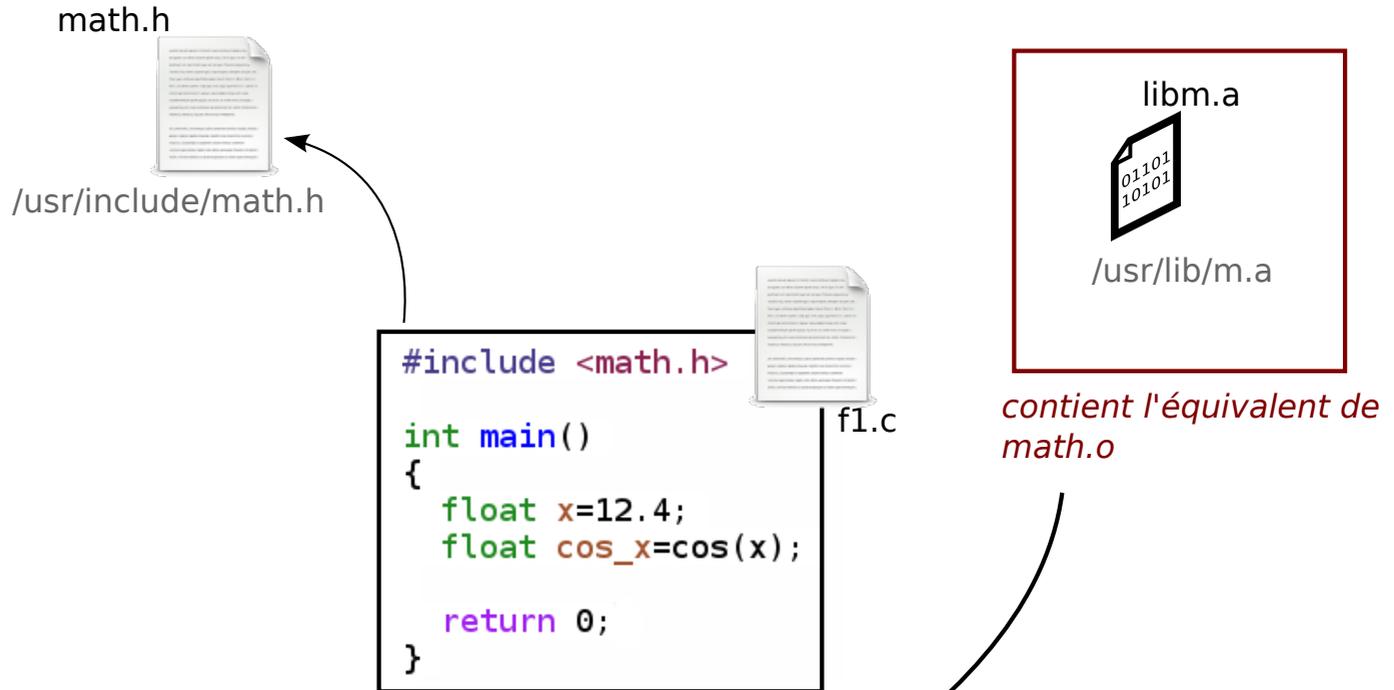
```
$ gcc -c f1.c -o f1.o -Wall -Wextra -g
> OK
```

```
$ gcc f1.o -o mon_executable
f1.o: In function `main':
f1.c:(.text+0x1a): undefined reference to `cos'
collect2: error: ld returned 1 exit status
```

*Edition de lien échoue*

Ne trouve pas  
l'**implémentation** de cos

# Edition de liens et bibliothèques



```
$ gcc -c f1.c -o f1.o -Wall -Wextra -g  
> OK
```

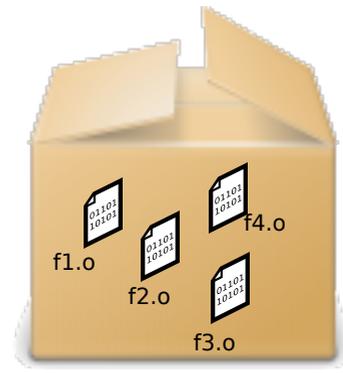
```
$ gcc f1.o -o mon_executable -lm  
> OK
```

se lier à libm

# Edition de liens et bibliothèques

Librairie statique:

`lib<NOM>.a` équivalent à



*archive de fichiers objets*

Lors de l'édition de liens:

`gcc fichier.o -l<NOM>` équivalent à `gcc fichier.o f1.o f2.o f3.o f4.o`

=> Inclusion des binaires dans l'executable finale

# Edition de liens et bibliothèques

Exemple de bibliothèque:

*bibliothèque vecteur:*

```
#ifndef INCLUDE_GUARD_V3_H
#define INCLUDE_GUARD_V3_H

struct v3
{
    float x,y,z;
};

void v3_init(struct v3* vec);

#endif
```



v3.h

```
#include "v3.h"

void v3_init(struct v3* vec)
{
    vec->x=0.0;
    vec->y=0.0;
    vec->z=0.0;
}
```



v3.c

*programme utilisant la bibliothèque:*

```
#include "v3.h"

int main()
{
    struct v3 mon_vecteur;
    void v3_init(&mon_vecteur);

    return 0;
}
```



main.c

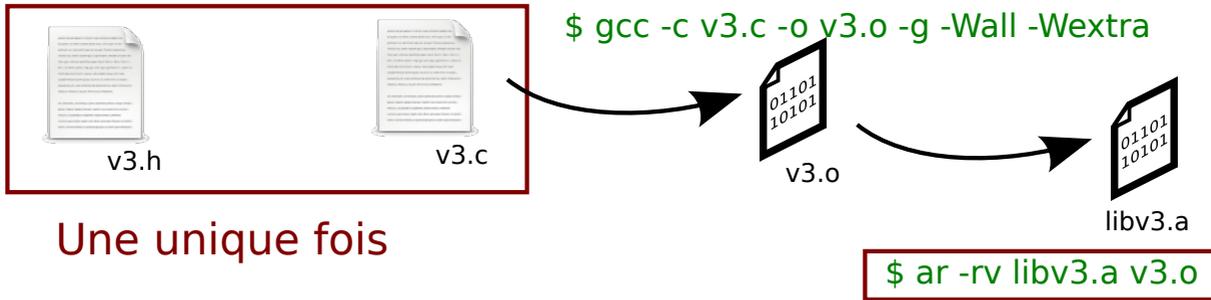
# Edition de liens et bibliothèques

Exemple de librairie:

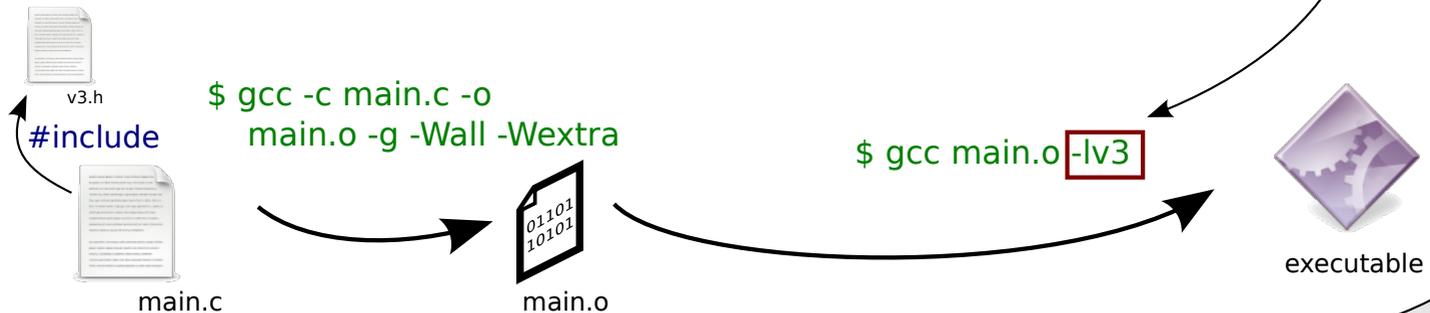
```
librairie vecteur:
#include <stdio.h>
#include <stdlib.h>
int v3;
void v3_init(v3* v) {
    *v = 0;
}
void v3_increment(v3* v) {
    *v++;
}

programme utilisant la librairie:
#include "v3.h"
int main() {
    v3 v;
    v3_init(&v);
    v3_increment(&v);
    return 0;
}
```

## 1. Création de la librairie:



## 2. Utilisation de la librairie:



# Edition de liens et bibliothèques

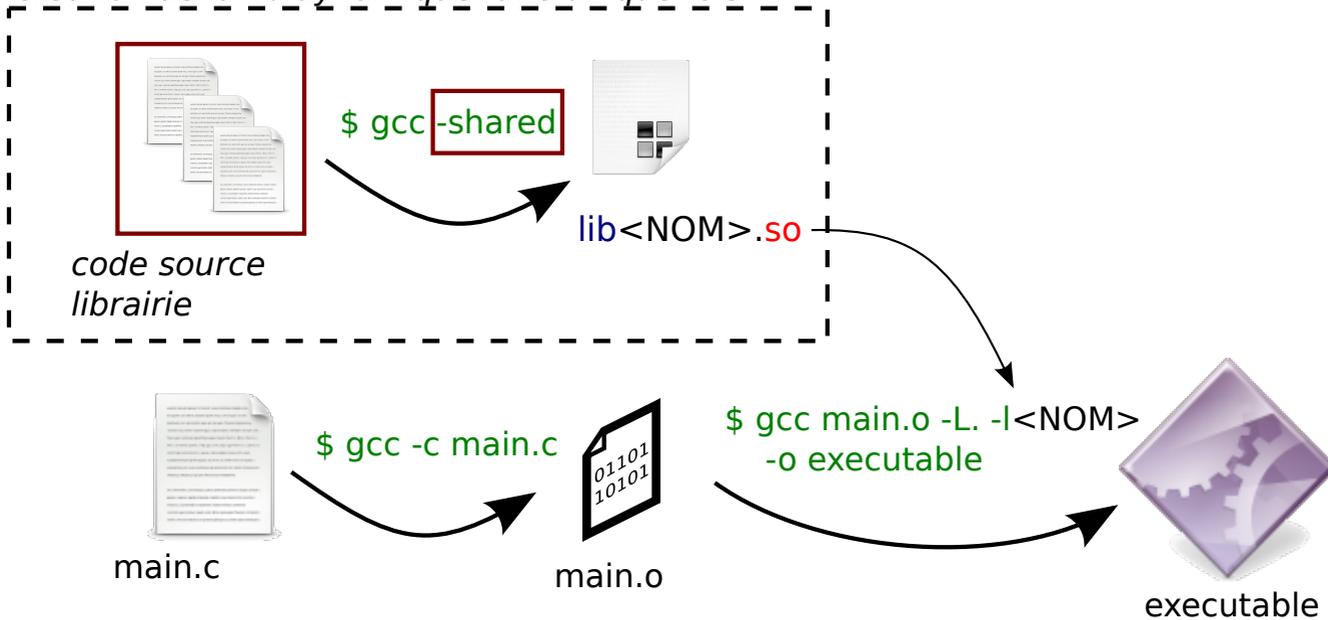
Librairies dynamiques:



lib<NOM>.so

Sont chargées dynamiquement à **chaque exécution**

*creation de la lib dynamique: une unique fois*



# Edition de liens et bibliothèques

Librairies dynamiques:



lib<NOM>.so

Sont chargées dynamiquement à **chaque exécution**

Lors de l'exécution:

\$ ./executable



executable



lib<NOM>.so

viens lire  
et charger le contenu de la lib

# Edition de liens et bibliothèques

Librairies dynamiques:



lib<NOM>.so

Sont chargées dynamiquement à **chaque exécution**

**Remarques:**



1. Si la lib change => l'executable change de comportement  
**Sans avoir à être recompilée**

*(update, MAJ, comportement, ...)*

2. Si la lib disparaît/est corrompu/n'est pas trouvée  
=> l'executable ne peut pas être lancé

# Edition de liens et bibliothèques

Exemple concret de bibliothèques dynamiques:

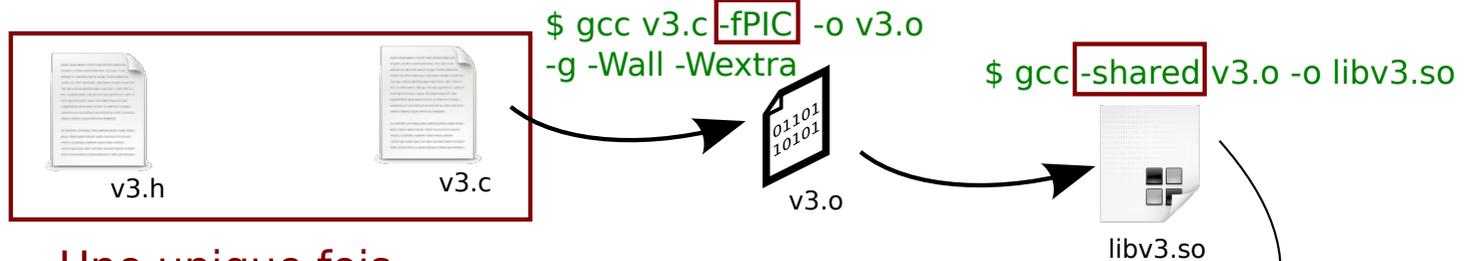


```
librairie vecteur:
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int x,y,z;
    void v3_init(int *v);
    void v3_init_vecteur(int *v);
    return 0;
}
v3.c

#include "v3.h"
void v3_init(int *v)
{
    *v = 0;
}
v3.h

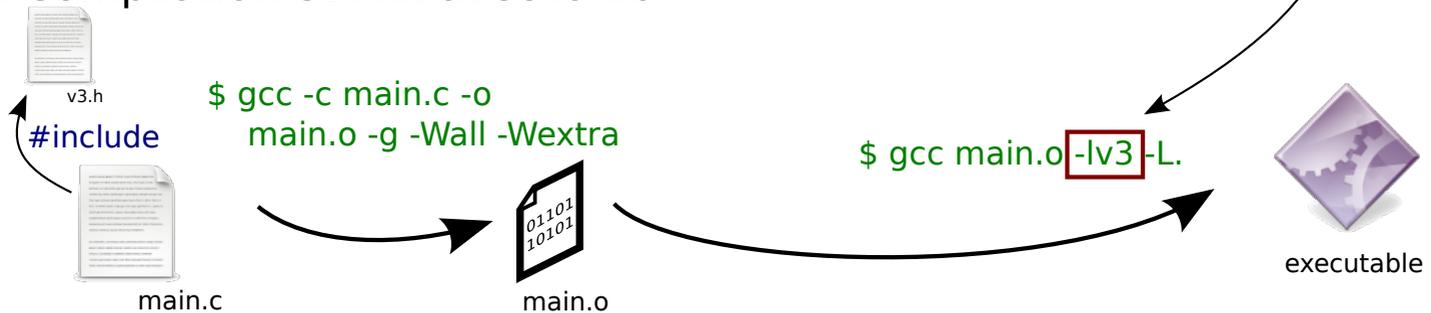
#include "v3.h"
int main()
{
    int x,y,z;
    void v3_init(int *v);
    void v3_init_vecteur(int *v);
    return 0;
}
main.c
```

## 1. Création de la librairie:



Une unique fois

## 2. Compilation et link avec la lib



# Edition de liens et librairies



Exemple concret de librairies dynamiques:

```
librairie vecteur:
#include <stdio.h>
#include <stdlib.h>
int main()
{
    float x,y,z;
}
void v3_init(float *v)
{
    *v=0;
}
void v3_init(float *v)
{
    *v=0;
}
}
}

programme utilisant la librairie:
#include "v3.h"
int main()
{
    float x,y,z;
    v3_init(&x);
    v3_init(&y);
    v3_init(&z);
    return 0;
}
main.c
```



Utilisation de la librairie:



Par défaut: viens chercher la lib dans /usr/lib/

repertoires déjà enregistrés



\$ export LD\_LIBRARY\_PATH=\$LD\_LIBRARY\_PATH:.  
ajout du repertoire local (.)

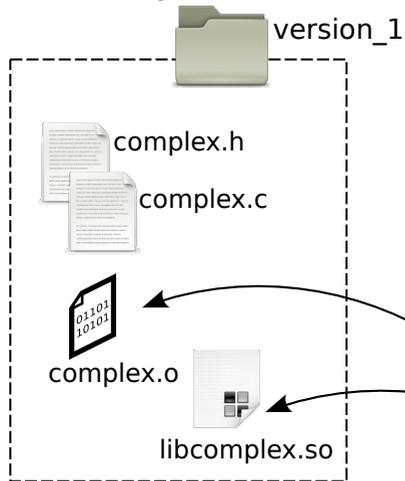
\$ ./executable

repertoire courant



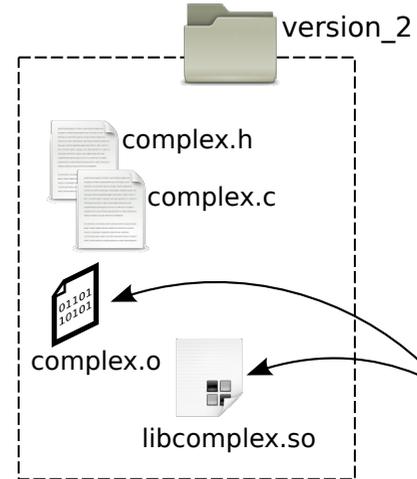
# Edition de liens et bibliothèques

Exemple:



```
$ gcc -c complex.c  
-g -Wall -Wextra -fPIC -o complex.o
```

```
$ gcc -shared complex.o -o libcomplex.so
```

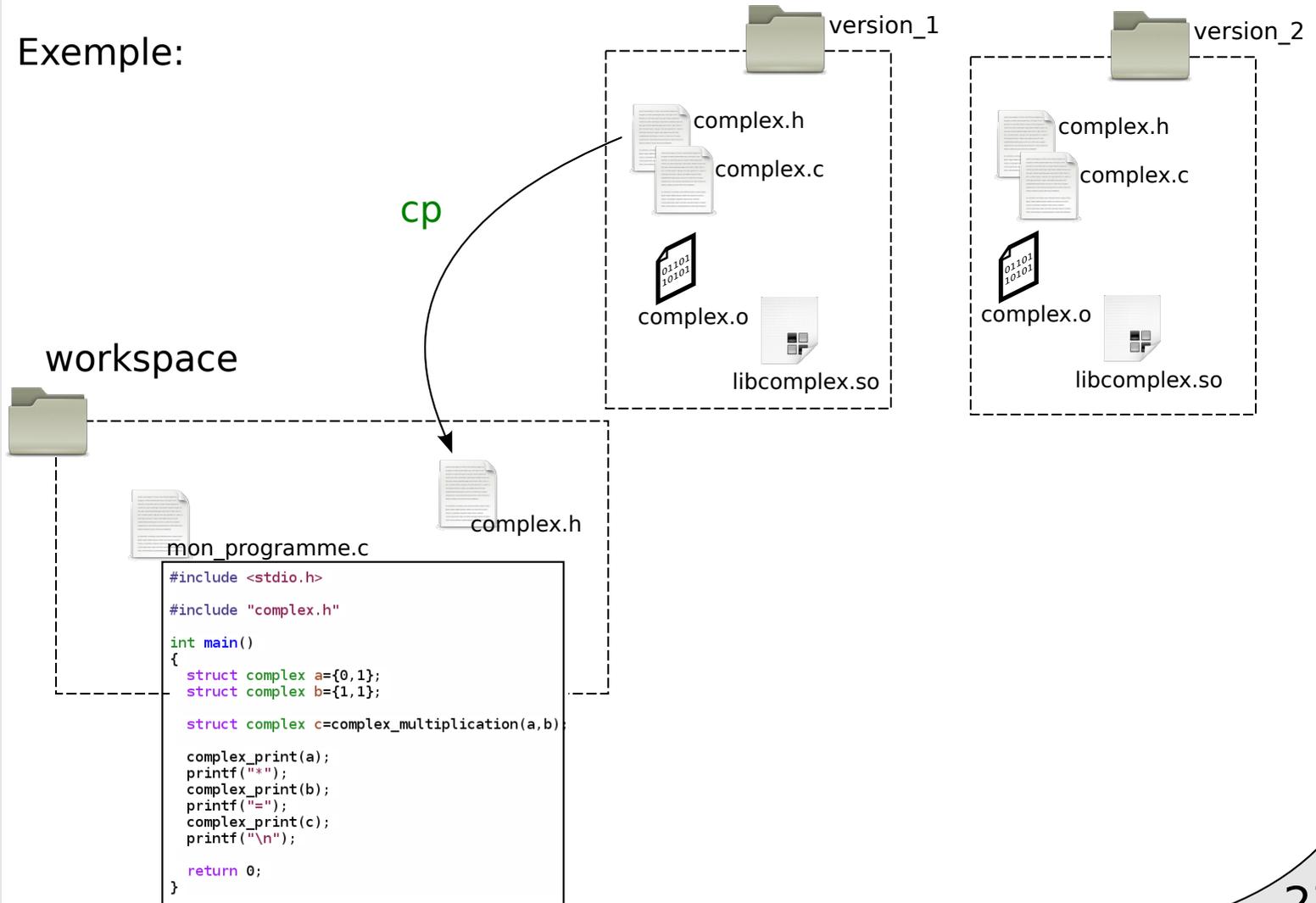


```
$ gcc -c complex.c  
-g -Wall -Wextra -fPIC -o complex.o
```

```
$ gcc -shared complex.o -o libcomplex.so
```

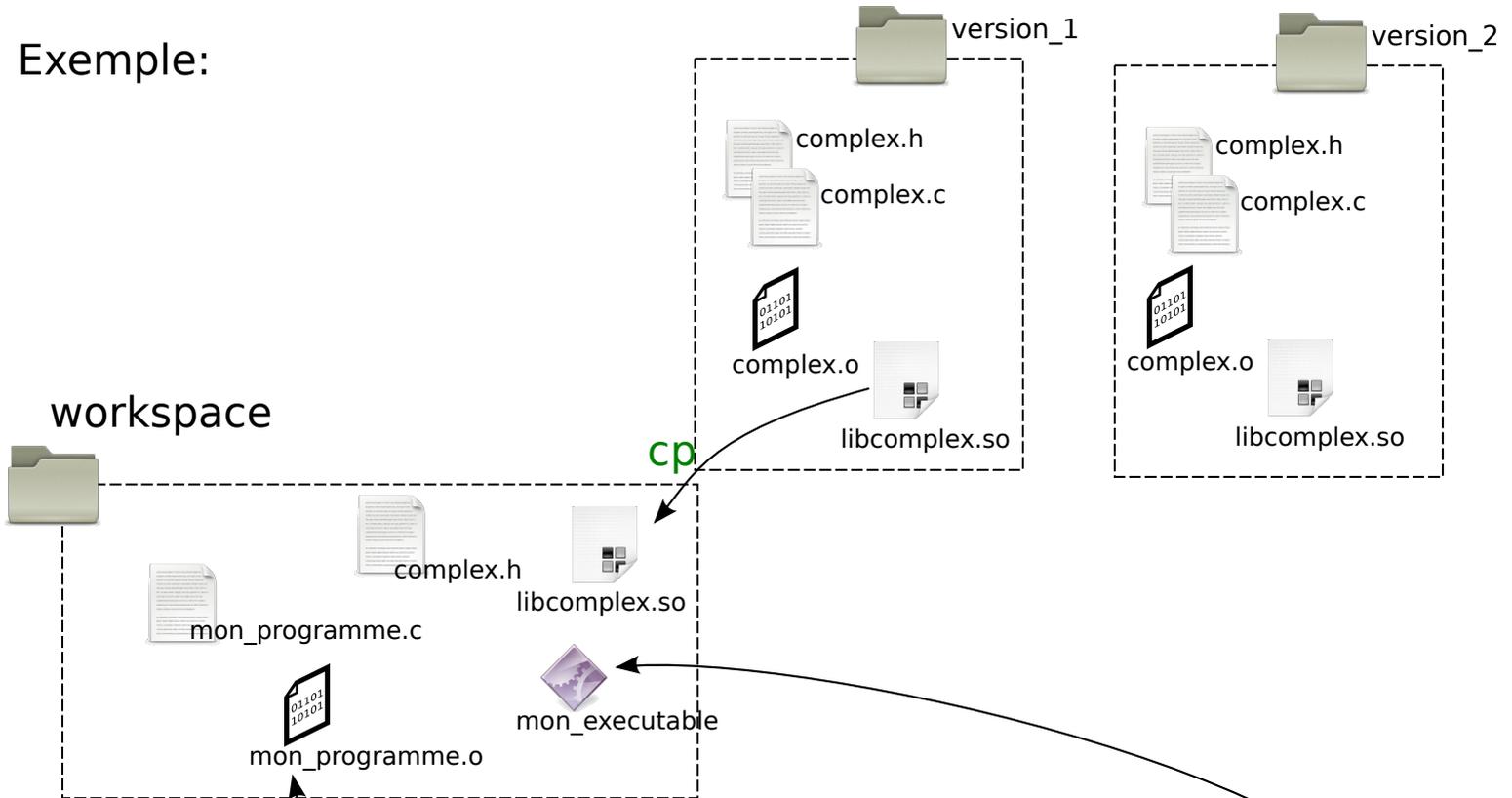
# Edition de liens et bibliothèques

Exemple:



# Edition de liens et bibliothèques

Exemple:

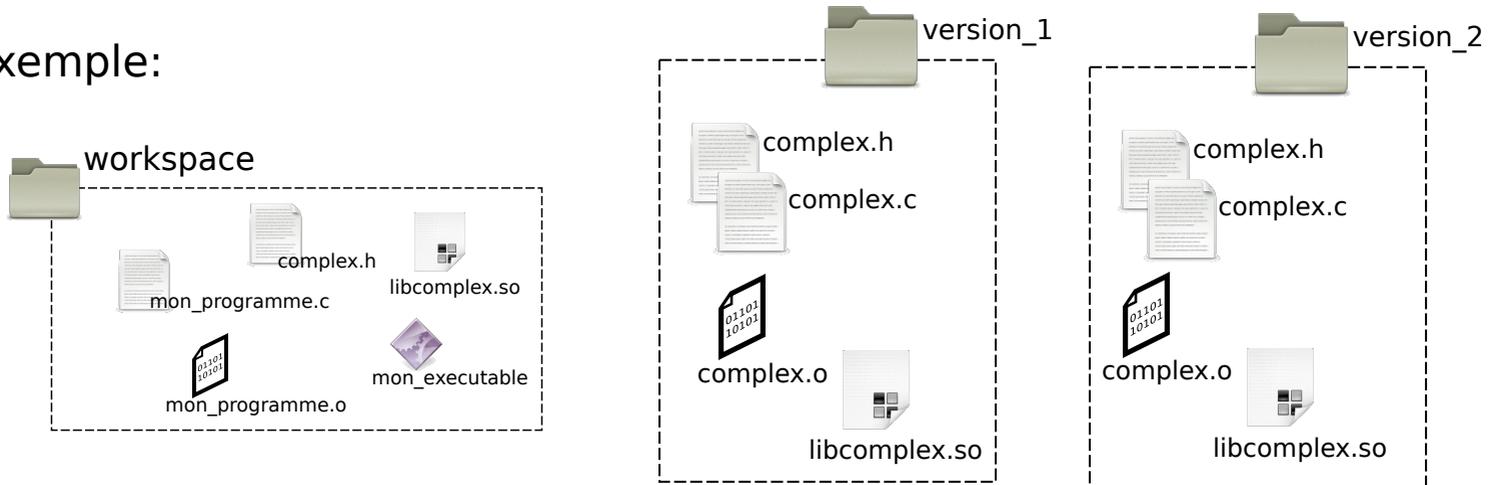


```
$ gcc -c mon_programme.c -g -Wall -Wextra
```

```
$ gcc mon_programme.o -L. -lcomplex -o mon_executable
```

# Edition de liens et bibliothèques

Exemple:



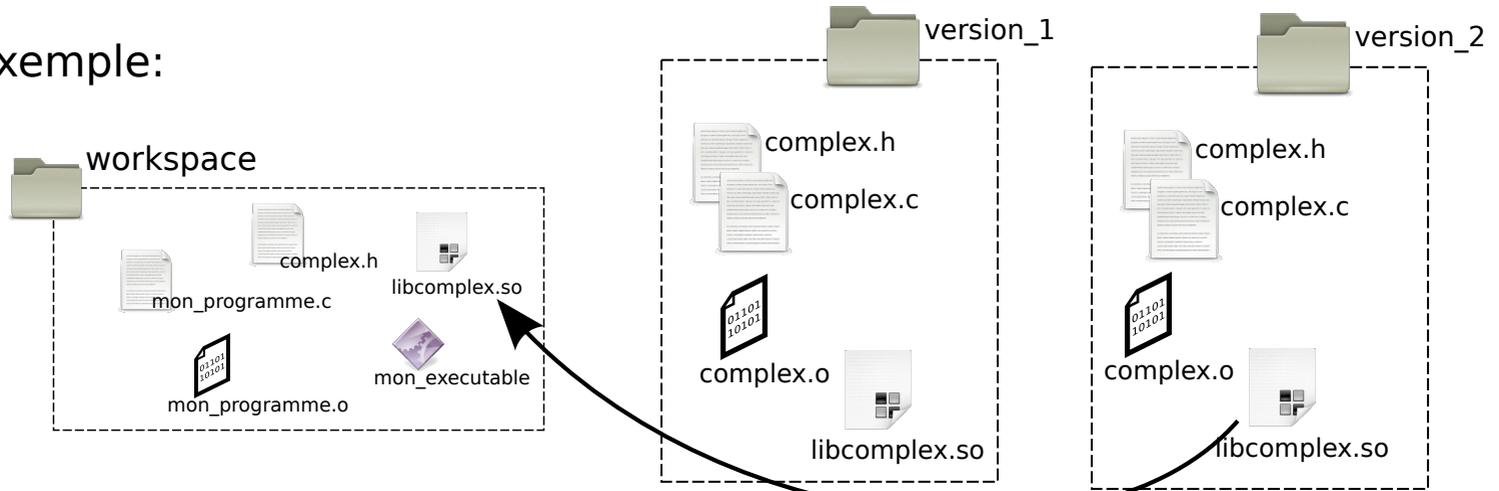
```
$ export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:.
```

```
$ ./mon_executable
```

```
(0.000000+1.000000 i)*(1.000000+1.000000 i)=(1.000000+1.000000 i)
```

# Edition de liens et bibliothèques

Exemple:



cp

(erase version\_1)

```
$ export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:.
```

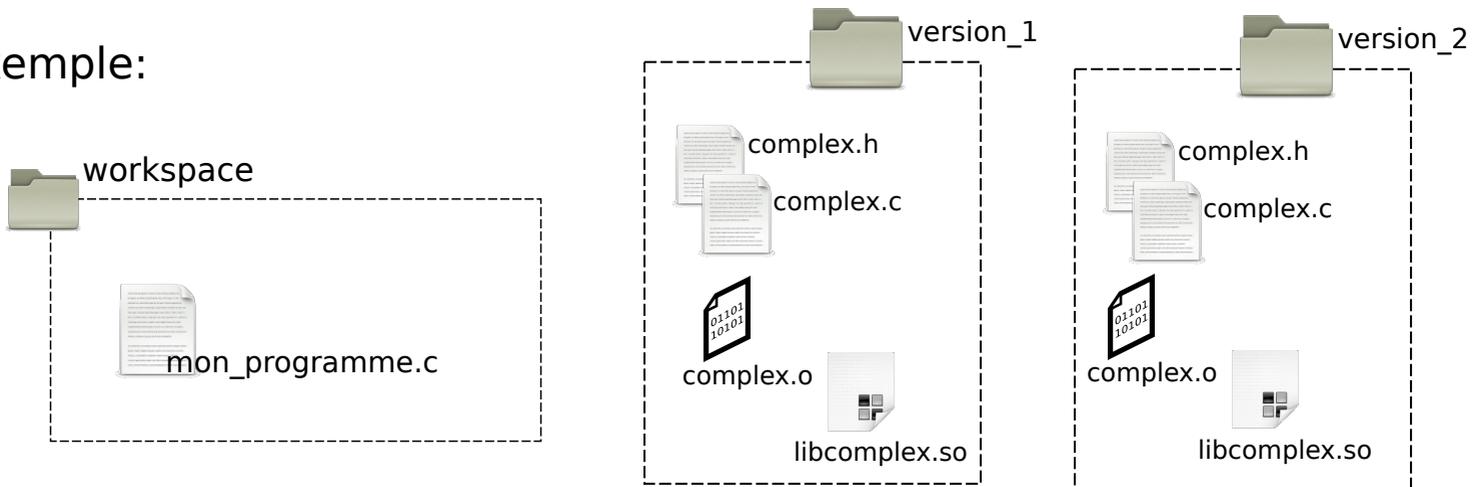
```
$ ./mon_executable
```

```
(0.000000+1.000000 i)*(1.000000+1.000000 i)=(-1.000000+1.000000 i)
```

*(principe du patch logiciel)*

# Edition de liens et bibliothèques

Exemple:



Si on ne veut pas copier/déplacer de fichiers:



```
$ gcc -c mon_programme.c -I version_1/ -g -Wall -Wextra
```



`mon_programme.o`

```
$ gcc mon_programme.o -L version_1/ -l complex -o executable
```



`executable`

```
$ export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:version_1/  
$ ./executable
```

# Entrées/sorties

Affichage écran

Chaine de caractères: stockage, bonnes pratiques

Ecriture/Lecture texte

Ecriture/Lecture fichiers (ASCII)

# Entrees/sorties

Affichage/lecture écran/fichier

## **Ecran:**

Information utilisateur

Debug d'un programme

Communication avec l'utilisateur

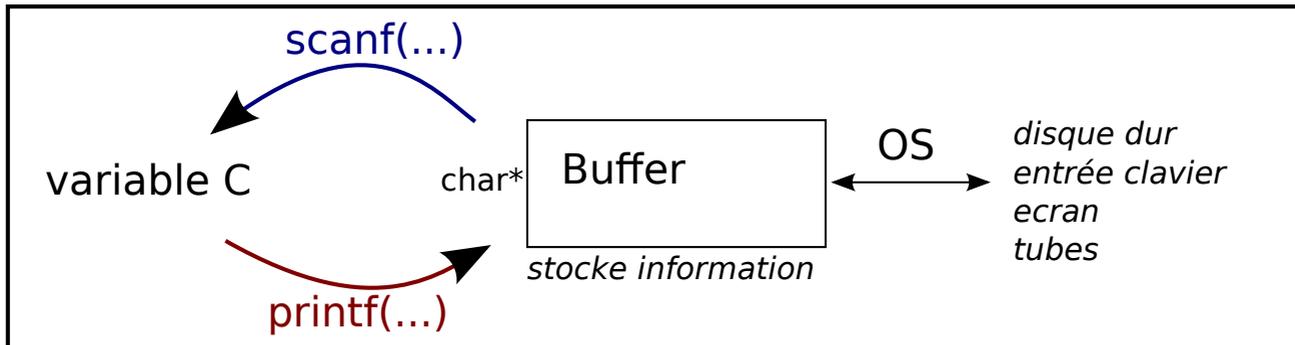
## **Fichier:**

Suivit execution (fichier log)

Sauvegarde information (disque=mémoire non volatile)

Communication entre programme (disque=mémoire partagée)

*En C: écriture fichier // écriture écran*



# Entrees/sorties

Rappels caractère *char*

Un caractère (char) = nombre entre 0-256

1 octet  
= 1 emplacement mémoire  
= 2 caractères hexa

ex.

```
char c='M';
```

4D

```
int value=(int)c;
```

↖ 77

Dec	Hx	Oct	Chr	Dec	Hx	Oct	Htm	Chr	Dec	Hx	Oct	Htm	Chr	Dec	Hx	Oct	Htm	Chr
0	0	000	NUL (null)	32	20	040	€#32;	Space	64	40	100	€#64;	0	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	EOT (end of transmission)	36	24	044	€#36;	€	68	44	104	€#68;	D	100	64	144	€#100;	d
5	5	005	ENO (enquiry)	37	25	045	€#37;	€	69	45	105	€#69;	E	101	65	145	€#101;	e
6	6	006	ACK (acknowledge)	38	26	046	€#38;	€	70	46	106	€#70;	F	102	66	146	€#102;	f
7	7	007	BEL (bell)	39	27	047	€#39;	€	71	47	107	€#71;	G	103	67	147	€#103;	g
8	8	010	BS (backspace)	40	28	050	€#40;	(	72	48	110	€#72;	H	104	68	150	€#104;	h
9	9	011	TAB (horizontal tab)	41	29	051	€#41;	)	73	49	111	€#73;	I	105	69	151	€#105;	i
10	A	012	LF (NL line feed, new line)	42	2A	052	€#42;	*	74	4A	112	€#74;	J	106	6A	152	€#106;	j
11	B	013	VT (vertical tab)	43	2B	053	€#43;	+	75	4B	113	€#75;	K	107	6B	153	€#107;	k
12	C	014	FF (NP form feed, new page)	44	2C	054	€#44;	,	76	4C	114	€#76;	L	108	6C	154	€#108;	l
13	D	015	CR (carriage return)	45	2D	055	€#45;	-	77	4D	115	€#77;	M	109	6D	155	€#109;	m
14	E	016	SO (shift out)	46	2E	056	€#46;	.	78	4E	116	€#78;	N	110	6E	156	€#110;	n
15	F	017	SI (shift in)	47	2F	057	€#47;	/	79	4F	117	€#79;	O	111	6F	157	€#111;	o
16	10	020	DLE (data link escape)	48	30	060	€#48;	0	80	50	120	€#80;	P	112	6F	158	€#112;	p
17	11	021	DC1 (device control 1)	49	31	061	€#49;	1	81	51	121	€#81;	Q	113	71	161	€#113;	q
18	12	022	DC2 (device control 2)	50	32	062	€#50;	2	82	52	122	€#82;	R	114	72	162	€#114;	r
19	13	023	DC3 (device control 3)	51	33	063	€#51;	3	83	53	123	€#83;	S	115	73	163	€#115;	s
20	14	024	DC4 (device control 4)	52	34	064	€#52;	4	84	54	124	€#84;	T	116	74	164	€#116;	t
21	15	025	NAK (negative acknowledge)	53	35	065	€#53;	5	85	55	125	€#85;	U	117	75	165	€#117;	u
22	16	026	SYN (synchronous idle)	54	36	066	€#54;	6	86	56	126	€#86;	V	118	76	166	€#118;	v
23	17	027	ETB (end of trans. block)	55	37	067	€#55;	7	87	57	127	€#87;	W	119	77	167	€#119;	w
24	18	030	CAN (cancel)	56	38	070	€#56;	8	88	58	130	€#88;	X	120	78	170	€#120;	x
25	19	031	EM (end of medium)	57	39	071	€#57;	9	89	59	131	€#89;	Y	121	79	171	€#121;	y
26	1A	032	SUB (substitute)	58	3A	072	€#58;	:	90	5A	132	€#90;	Z	122	7A	172	€#122;	z
27	1B	033	ESC (escape)	59	3B	073	€#59;	;	91	5B	133	€#91;	[	123	7B	173	€#123;	{
28	1C	034	FS (file separator)	60	3C	074	€#60;	<	92	5C	134	€#92;	\	124	7C	174	€#124;	
29	1D	035	GS (group separator)	61	3D	075	€#61;	=	93	5D	135	€#93;	]	125	7D	175	€#125;	}
30	1E	036	RS (record separator)	62	3E	076	€#62;	>	94	5E	136	€#94;	^	126	7E	176	€#126;	~
31	1F	037	US (unit separator)	63	3F	077	€#63;	?	95	5F	137	€#95;	_	127	7F	177	€#127;	DEL

Caractère: entre ' '

# Entrées/sorties

Affichage écran

→ **Chaine de caractères: stockage, bonnes pratiques**

Ecriture/Lecture texte

Ecriture/Lecture fichiers (ASCII)

# Entrees/sorties

Rappels chaine de caractère

*const char\** , *char []*

Chaine de caractère = suite de caractères

**char** nom[10]; → reserve 10 emplacements mémoire

nom[0]='m';

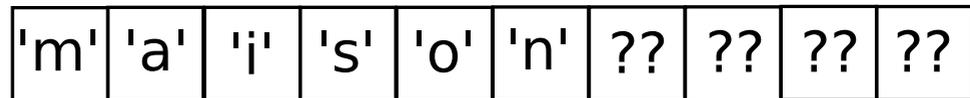
nom[1]='a';

nom[2]='i';

nom[3]='s';

nom[4]='o';

nom[5]='n';



↓ en mémoire

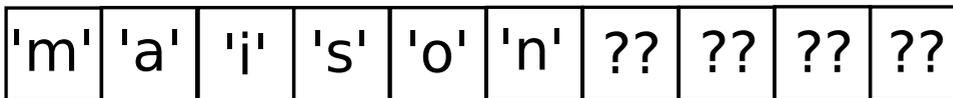


# Entrees/sorties

Rappels chaine de caractère

`const char*` , `char []`

Chaine de caractère = suite de caractères



↓ en mémoire



`nom` correspond à la première case de la chaine

`nom` = pointeur sur la 1ere case

```
#include <stdio.h>

int main()
{
    char nom[10];
    nom[0]='m';
    nom[1]='a';
    nom[2]='i';
    nom[3]='s';
    nom[4]='o';
    nom[5]='n';

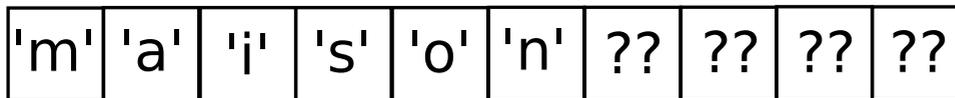
    printf("%s\n",nom);
    return 0;
}
```

# Entrees/sorties

Rappels chaine de caractère

`const char*` , `char []`

Chaine de caractère = suite de caractères



↓ en mémoire



Comment savoir où s'arrêter ?

```
#include <stdio.h>

int main()
{
    char nom[10];
    nom[0]='m';
    nom[1]='a';
    nom[2]='i';
    nom[3]='s';
    nom[4]='o';
    nom[5]='n';

    printf("%s\n",nom);
    return 0;
}
```

# Entrees/sorties

Rappels chaine de caractere

`const char*` , `char []`

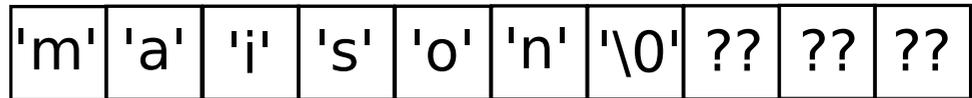


Fin de chaine = principe de **sentinelle de fin**

```
#include <stdio.h>

int main()
{
    char nom[10];
    nom[0]='m';
    nom[1]='a';
    nom[2]='i';
    nom[3]='s';
    nom[4]='o';
    nom[5]='n';
    nom[6]='\0';

    printf("%s\n", nom);
    return 0;
}
```



↓ en memoire



Affiche **maison**

Comportement deterministe

# Entrees/sorties

Rappels chaine de caractere

`const char* , char []`

Fin de chaine = `'\0'`

Toujours



Toujours

=> Toujours prévoir une case pour placer `'\0'`

Toujours

Fonction pour trouver la taille d'un mot:

```
int calcul_taille(char mot[])
{
    char *pointeur_debut=mot;
    char *pointeur_fin=pointeur_debut;

    while(*pointeur_fin!='\0')
        ++pointeur_fin;

    return (int)(pointeur_fin-pointeur_debut);
}
```

# Entrees/sorties

Rappels chaine de caractere

*const char\** , *char []*

Fonction pour trouver la taille d'un mot:

```
int calcul_taille(char mot[])
{
    char *pointeur_debut=mot;
    char *pointeur_fin=pointeur_debut;

    while(*pointeur_fin!='\0')
        ++pointeur_fin;

    return (int)(pointeur_fin-pointeur_debut);
}
```

**Attention!! Si '\0' est oublié !!!**

=> Comportement indetermine X

# Entrees/sorties

Rappels chaine de caractere

`const char*` , `char []`

Les fonctions de manipulation de chaine: `#include <string.h>`

`str<...>`

`strcpy`  
`strcmp`  
`strcat`  
...

*jusqu'à '\0'*

`strncpy`  
`strncmp`  
`strncat`  
...

*jusqu'à '\0'*  
*maximum `n` caractères*

→ copie  
→ compare  
→ concatenation

# Bonnes pratiques: Chaîne de caractères

Définir les chaînes "inline" en tant que `char[]` `char[]="ma_chaine"`

Définir les pointeurs (`char*`) en tant que constantes  
`const char*`

Attention:

```
char mot[]="abricot";  
mot[2]='l';
```

OK



```
char *mot="abricot";  
mot[2]='l';
```

KO



pointeur vers  
une chaîne constante

# Bonnes pratiques: Chaîne de caractères

Toujours utiliser les fonctions à tailles limitées:

`strn<...>`

~~`strcpy(mot_1,mot_2);  
strcmp(mot_1,mot_2);  
strcat(mot_1,mot_2);`~~

`strncpy(mot_1,mot_2,n_max);  
strncmp(mot_1,mot_2,n_max);  
strncat(mot_1,mot_2,n_max);`

+ sécurisé

+ debug plus aisé

## **Note:**

*Le C n'est pas le langage approprié pour le traitement complexe de chaîne de caractères*

# Bonnes pratiques: Chaîne de caractères

Ne pas utiliser d'opérateurs sur des chaîne de caractères !!!



```
#include <stdio.h>
int main()
{
    char mot_1[]="abricot";
    char *mot_2="peche";

    mot_2=mot_1;
    printf ("%s\n", mot_2);

    char mot_3[]="abricot";
    if(mot_1==mot_3)
        printf("meme mot");

    char* mot_4="hello ";
    char* mot_5="world";

    mot_4 += *mot_5;

    return 0;
}
```

Attention!!!

gcc -Wall -Wextra

ne donne aucun Warning !

Pourtant ce programme ne fait  
probablement pas ce que vous souhaitez!

# Bonnes pratiques: Chaîne de caractères

Ne pas utiliser d'opérateurs sur des chaîne de caractères !!!

```
#include <stdio.h> X

int main()
{
    char mot_1[]="abricot";
    char *mot_2="peche";

    mot_2=mot_1;
    printf ("%s\n",mot_2);

    char mot_3[]="abricot";
    if(mot_1==mot_3)
        printf("meme mot");

    char* mot_4="hello ";
    char* mot_5="world";

    mot_4 += *mot_5;

    return 0;
}
```

OK

mauvaise habitude

affectation de pointeurs !!  
ce n'est pas une copie de chaîne !

mauvaise habitude

test d'égalité d'adresse de pointeur!  
ne compare pas la chaîne (ici test = faux)

mauvaise habitude

mauvaise habitude

ajoute (int)(mot\_5[0])=119 à l'adresse de mot\_4  
Ne réalise absolument la concaténation d'une chaîne!!

# Bonnes pratiques: Chaîne de caractères

Ne pas utiliser d'opérateurs sur des chaînes de caractères !!!



```
#include <stdio.h>

int main()
{
    char mot_1[]="abricot";
    char *mot_2="peche";

    mot_2=mot_1;
    printf("%s\n", mot_2);

    char mot_3[]="abricot";
    if(mot_1==mot_3)
        printf("meme mot");

    char* mot_4="hello ";
    char* mot_5="world";

    mot_4 += *mot_5;

    return 0;
}
```

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

#define N 10

int main()
{
    char mot_1[N]="abricot";
    char mot_2[N]="peche";

    strncpy(mot_1, mot_2, N);

    char mot_3[N]="abricot";
    if(strncmp(mot_1, mot_3, N)==0)
        printf("meme mot");

    char mot_4[N]="hello";
    char mot_5[N]=" world";
    strncat(mot_4, mot_5, N);

    return 0;
}
```

**OK**

# Entrées/sorties

Affichage écran

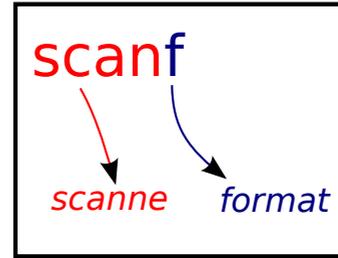
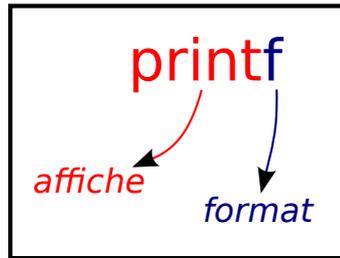
Chaine de caractères: stockage, bonnes pratiques

→ **Ecriture/Lecture texte**

Ecriture/Lecture fichiers (ASCII)

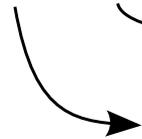
# Printf / Scanf

## Fonctions de conversions



## Principe:

**a=printf**(<format>, variables ...)

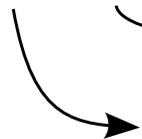


renvoie le nombre de caractères formatés

→ affiche sur la sortie standard (stdout)

*en général: la ligne de commande écran*

**a=scanf**(<format>, variables ...)



renvoie le nombre d'arguments formatés

→ récupère de l'entrée standard (stdin)

*en général: la ligne de commande clavier*

# Printf / Scanf

ex

```
int a=12;
printf("%d",a); //affiche: 12

int b=0xA AFF4E3D;
printf("%x",b); //affiche aaff4e3d

char c='e';
printf("%c",c); //affiche: e

char mot[]="bonjour a tous";
printf("%s",mot); //affiche: bonjour a tous

float x=1.25;
printf("%f",x); //affiche 1.250000
printf("%.3f",x); //affiche 1.250
```

# Printf / Scanf

ex

printf (et scanf) accepte un nombre d'argument variable  
=> Fonction variadiques

```
printf("\n");  
printf("Je suis %s a %c%c%c \n", "etudiant", 'C', 'P', 'E');  
printf("j'ai %d ans\n", 15);  
  
printf("1.3+1.7=%1.2f\n", 1.3+1.7);
```

```
int a=printf("\n");  
int b=printf("Je suis %s\n", "heureux");  
int c=printf("%d-%d=%d\n", 5, 7, 5-7);  
int d=printf("%d+4=%d\n", 3);  
  
printf("%d %d %d %d\n", a, b, c, d);
```

```
Je suis heureux  
5-7=-2  
3+4=1522178016  
1 16 7 15
```

oublie argument  
=> comportement indeterminé  
(=> Warnings!)

# Printf / Scanf

ex

```
int main()
{
    int a=0;
    scanf ("%d", &a);

    float x=0.0;
    scanf ("%f", &x);

    char c='a';
    scanf (" %c", &c);

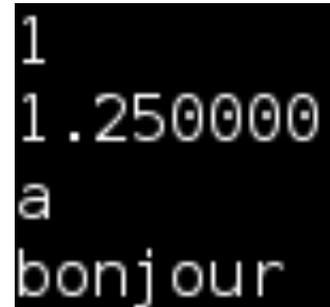
    char buffer[50];
    scanf ("%50s", buffer);

    printf ("%d\n%f\n%c\n%s\n", a, x, c, buffer);

    return 0;
}
```

```
$ ./mon_executable
> 1
> 1.25      entrées
> a         utilisateur
> bonjour
```

affiche:



```
1
1.250000
a
bonjour
```

# Printf / Scanf

ex

```
int main()
{
    int a=0;
    scanf ("%d", &a);

    float x=0.0;
    scanf ("%f", &x);

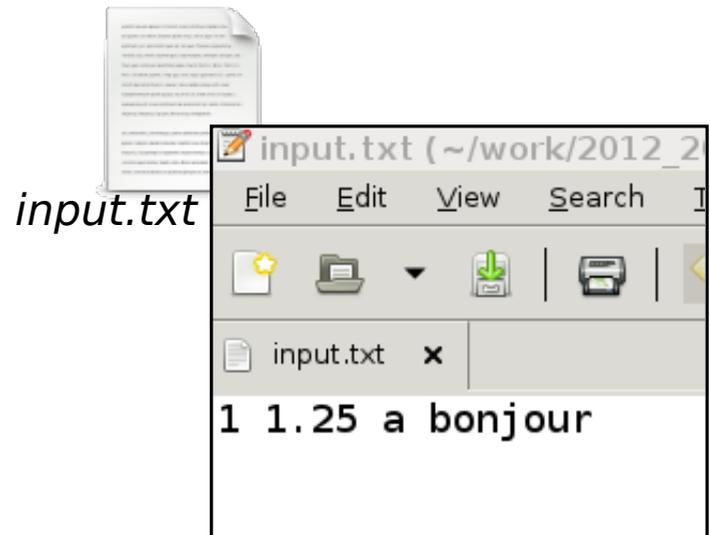
    char c='a';
    scanf (" %c", &c);

    char buffer[50];
    scanf ("%50s", buffer);

    printf ("%d\n%f\n%c\n%s\n", a, x, c, buffer);

    return 0;
}
```

Astuce:  
entrée utilisateur:



```
$ ./mon_executable < input.txt
```

```
1
1.250000
a
bonjour
```

l'entrée standard devient le fichier!  
=> scanf vient "lire" dans le fichier

# Printf / Scanf

ex

```
int main()
{
    int x=0,y=0,z=0;
    scanf("vecteur: %d %d %d\n", &x, &y, &z);

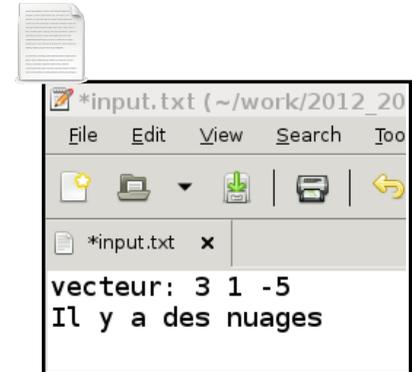
    char buffer[256];
    scanf("Il y a des %256s\n",buffer);

    printf("(%d,%d,%d) , %s\n",x,y,z,buffer);

    return 0;
}
```

```
$ ./mon_executable < input.txt
> (3,1,-5) , nuages
```

input.txt



# Printf / Scanf

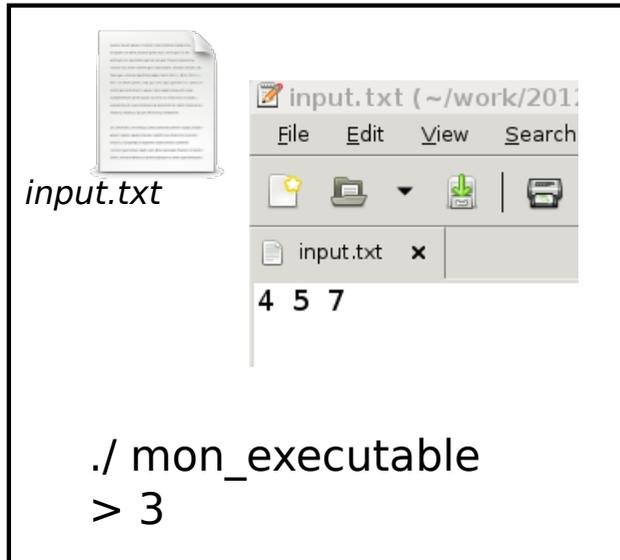
ex

```
int main()
{
    int x=0,y=0,z=0;

    int valeur=scanf ("%d %d %d",&x,&y,&z);

    printf ("%d\n",valeur);

    return 0;
}
```



input.txt

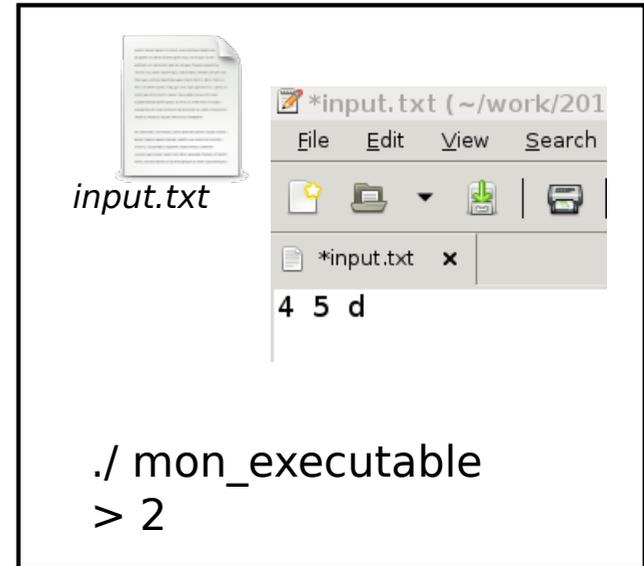
input.txt (~/work/201...)

File Edit View Search

input.txt x

4 5 7

./ mon\_executable  
> 3



input.txt

\*input.txt (~/work/201...)

File Edit View Search

\*input.txt x

4 5 d

./ mon\_executable  
> 2

# Printf / Scanf

Formatage sur d'autres entrées/sorties

**sprintf** / **sscanf**

string: chaîne de caractères

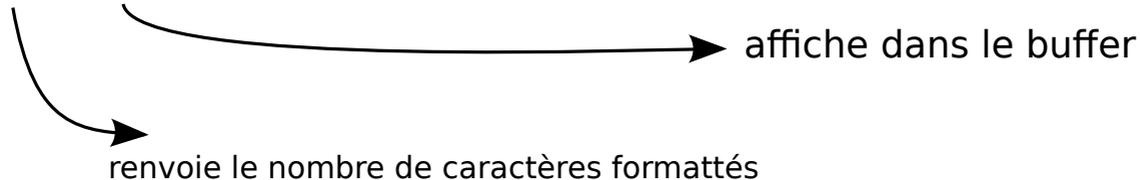


Principe:

`a=sprintf(buffer,<format>,variables ...)`

affiche dans le buffer

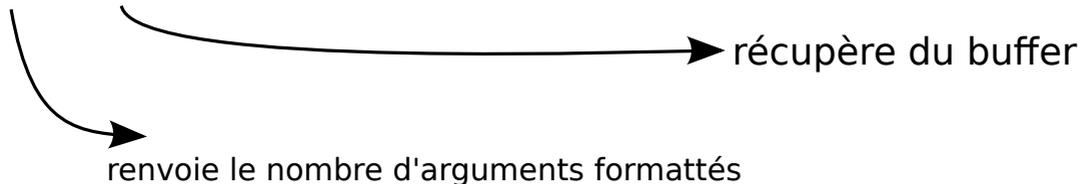
renvoie le nombre de caractères formatés



`a=sscanf(buffer,<format>,variables ...)`

récupère du buffer

renvoie le nombre d'arguments formatés



# Printf / Scanf

```
int main()
{
    char buffer[512];
    float x=cos(0.5);
    sprintf(buffer, "%s, (%d,%2.3f)", "bonjour", 1+4, x);

    printf("%s\n",buffer);

    return 0;
}
```

```
$ ./mon_executable
> bonjour, (5,0.878)
```

# Printf / Scanf

## Analyse d'une chaine de caractères

```
int main()
{
    char buffer[512]="Il fait beau ce matin. Il est 13 heures et fait 25.1 degres \n";

    char temps[25];
    int heure=0;
    float temperature=0.0;
    sscanf(buffer,"Il fait %s ce matin. Il est %d heures et fait %f degres \n",&temps,&heure,&temperature);

    printf("%s %d %1.2f\n", temps,heure,temperature);

    return 0;
}
```

```
$ ./mon_executable
> beau 13 25.10
```

# Entrées/sorties

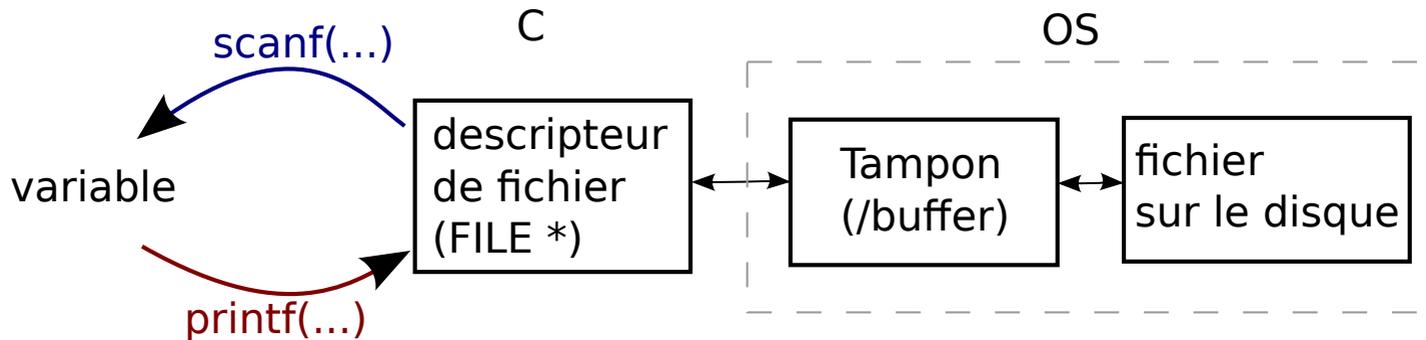
Affichage écran

Chaine de caractères: stockage, bonnes pratiques

Ecriture/Lecture texte

→ **Ecriture/Lecture fichiers (ASCII)**

# Ecriture/lecture fichier



## Ouvrir un fichier (pour l'écriture)

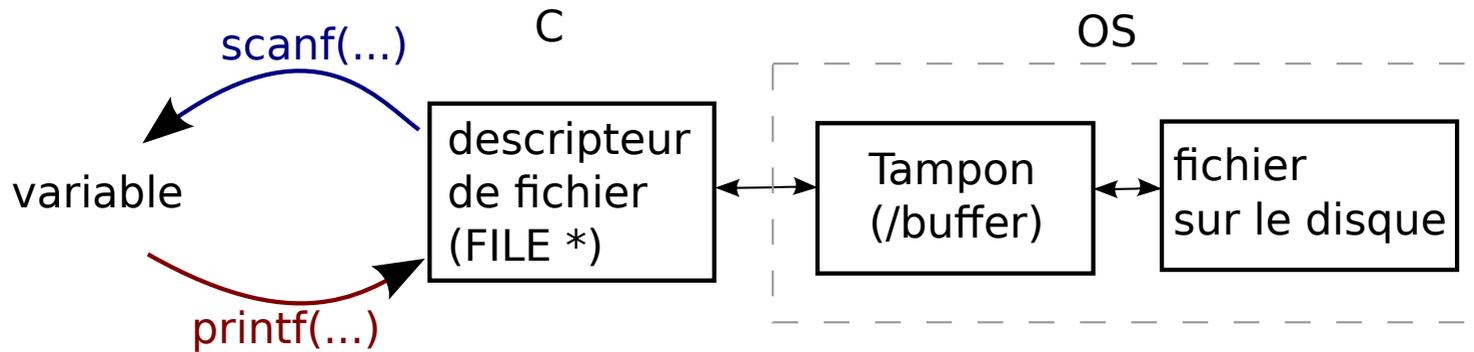
```
FILE* mon_descripteur=NULL; //pointeur vers descripteur de fichier
mon_fichier=fopen("mon_fichier.txt", "w"); //ouverture en écriture (w)
//créé le fichier si il n'existe pas
if(mon_fichier==NULL) //gestion d'erreur
{printf("Erreur ouverture fichier [mon_fichier.txt]\n");abort();}
```

## Ouvrir un fichier (pour la lecture)

```
FILE* mon_descripteur=NULL; //pointeur vers descripteur de fichier
mon_fichier=fopen("mon_fichier.txt", "r"); //ouverture en lecture (r)
//le fichier doit déjà exister
if(mon_fichier==NULL) //gestion d'erreur
{printf("Erreur ouverture fichier [mon_fichier.txt]\n");abort();}
```



# Ecriture/lecture fichier

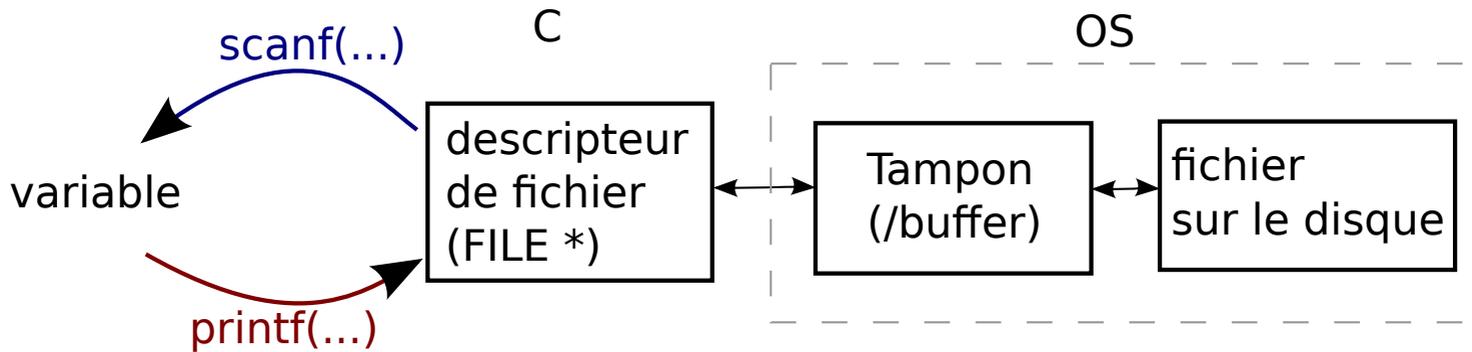


Fermer un fichier

```
int valeur=fopen(mon_descripteur); //ouverture fichier
if(valeur!=0) //gestion d'erreur
{printf("Erreur ouverture fichier \n");abort();}
```



# Ecriture/lecture fichier

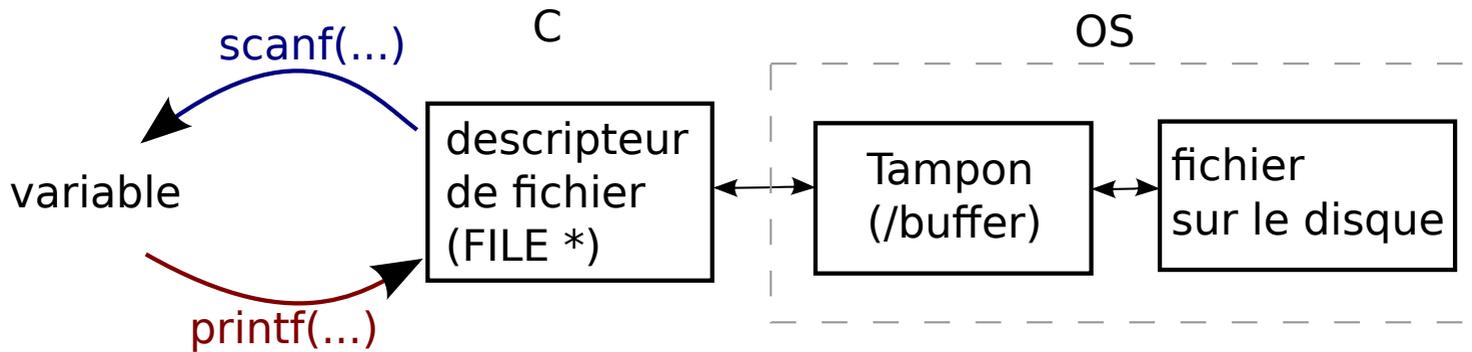


Ecrire dans un fichier *fprintf(...)*

```
int valeur=fopen(mon_descripteur, "%s", "Mon texte a moi\n");  
if (valeur!=1) //gestion d'erreur  
{printf("Erreur ecriture fichier\n"); abort();}
```



# Ecriture/lecture fichier



Lire dans un fichier *fscanf(...)*

```
char buffer[25]; //prepare un buffer de 25 cases
//lit au plus 25 caracteres
int valeur=fscanf(mon_descripteur, "%25s\n", buffer);
if(valeur!=1)//gestion d'erreur
{printf("Erreur lecture fichier\n");abort();}
```



# Ecriture/lecture fichier

Lecture/ecriture par `fprintf(...)` et `fscanf(...)`

Similaire à `printf(...)`, `scanf(...)`

↑  
écrit sur la sortie standard  
(`stdout => écran`)

↑  
lit sur l'entrée standard  
(`stdin => clavier`)

Exemples minimalistes fonctionnels :  
(Attention: sans gestion d'erreur)

```
#include <stdio.h>

int main()
{
    FILE *fid=fopen("mon_fichier.txt", "w");
    fprintf(fid, "prix carottes : 15 euros \n");
    fclose(fid);

    return 0;
}
```

*lecture*

```
#include <stdio.h>

int main()
{
    char legume[128];
    int prix=0;

    FILE *fid=fopen("mon_fichier.txt", "r");
    fscanf(fid, "prix %128s : %d euros \n", legume, &prix);
    fclose(fid);

    printf("les %s coutent %d euros\n", legume, prix);

    return 0;
}
```

*écriture*

# Ecriture/lecture fichier

Exemples minimalistes fonctionnels : Ecriture  
(Attention: sans gestion d'erreur)

```
#ifndef INCLUDE_GUARD_PERSONNE_H
#define INCLUDE_GUARD_PERSONNE_H

struct personne
{
    char nom[15];
    char fonction[15];
    int salaire;
    int anciennete;
};

#endif
```



*personne.h*

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

int main()
{
    //base de donnees
    struct personne employe[4]={{ "Charlie", "Directeur", 55000, 3},
                                  {"Robert", "DRH", 45000, 15},
                                  {"Brigitte", "R&D", 38000, 4},
                                  {"Raymond", "Technicien", 23000, 25}};

    //ouverture fichier
    FILE *fichier=fopen("ma_base.txt", "w");

    //ecriture fichier
    int k=0;
    for(k=0;k<4;++k)
    {
        fprintf(fichier, "%s %s %d %d \n", employe[k].nom,
                employe[k].fonction,
                employe[k].salaire,
                employe[k].anciennete);
    }

    //fermeture fichier
    fclose(fichier);

    return 0;
}
```



*ecrivain.h*

# Ecriture/lecture fichier

Exemples minimalistes fonctionnels : Ecriture  
(Attention: sans gestion d'erreur)

```
#ifndef INCLUDE_GUARD_PERSONNE_H
#define INCLUDE_GUARD_PERSONNE_H

struct personne
{
    char nom[15];
    char fonction[15];
    int salaire;
    int anciennete;
};

#endif
```



*personne.h*

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

int main()
{
    //base de donnees
    struct personne employe[4]={{ "Charlie", "Directeur", 55000, 3},
                                {"Robert", "DRH", 45000, 15},
                                {"Brigitte", "R&D", 38000, 4},
                                {"Raymond", "Technicien", 23000, 25}};

    //ouverture fichier
    FILE *fichier=fopen("ma_base.txt", "w");

    //ecriture fichier
    int k=0;
    for(k=0;k<4;++k)
    {
        fprintf(fichier, "%s %s %d %d \n", employe[k].nom,
                employe[k].fonction,
                employe[k].salaire,
                employe[k].anciennete);
    }

    //fermeture fichier
    fclose(fichier);

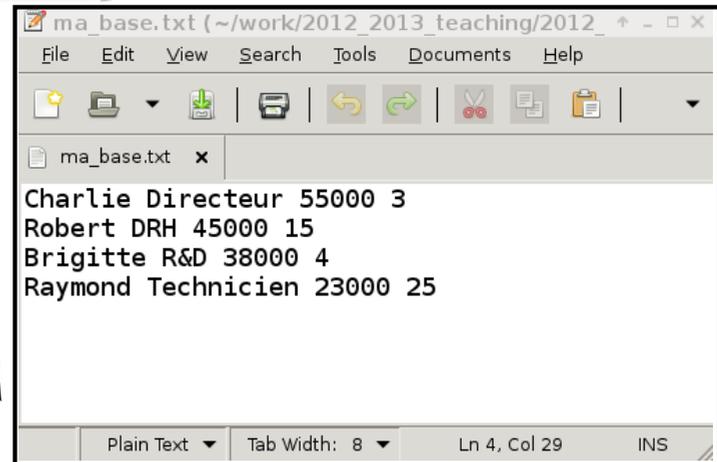
    return 0;
}
```



*ecrivain.h*



*ma\_base.txt*



```
ma_base.txt (~/.work/2012_2013_teaching/2012_ ...)
```

```
Charlie Directeur 55000 3
Robert DRH 45000 15
Brigitte R&D 38000 4
Raymond Technicien 23000 25
```

Plain Text Tab Width: 8 Ln 4, Col 29 INS

# Ecriture/lecture fichier

Exemples minimalistes fonctionnels : Lecture  
(Attention: sans gestion d'erreur)

```
#ifndef INCLUDE_GUARD_PERSONNE_H
#define INCLUDE_GUARD_PERSONNE_H

struct personne
{
    char nom[15];
    char fonction[15];
    int salaire;
    int anciennete;
};

#endif
```



*personne.h*

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

int main()
{
    //base de donnees
    struct personne employe[4];

    //ouverture fichier
    FILE *fichier=fopen("ma_base.txt", "r");

    //ecriture fichier
    int k=0;
    for(k=0;k<4;++k)
    {
        fscanf(fichier, "%15s %15s %d %d \n", employe[k].nom,
            employe[k].fonction,
            &employe[k].salaire,
            &employe[k].anciennete);
    }

    //fermeture fichier
    fclose(fichier);

    return 0;
}
```

# Ecriture/lecture fichier

## Exemple ecriture fichier avec gestion d'erreur



```
int main()
{
    //base de donnees
    struct personne employe[4]={{ "Charlie", "Directeur", 55000, 3},
                                  {"Robert", "DRH", 45000, 15},
                                  {"Brigitte", "R&D", 38000, 4},
                                  {"Raymond", "Technicien", 23000, 25}};

    //ouverture fichier
    char nom_fichier[]="ma_base.txt";

    FILE *fichier=NULL;
    fichier=fopen(nom_fichier,"w");
    if(fichier==NULL)
    {printf("Erreur ouverture fichier %s\n",nom_fichier);abort();}

    //ecriture fichier
    int k=0;
    for(k=0;k<4;++k)
    {
        int verif=fprintf(fichier,"%s %s %d %d \n",employe[k].nom,
                          employe[k].fonction,
                          employe[k].salaire,
                          employe[k].anciennete);

        if(verif<=0)
        {printf("Erreur ecriture fichier %s\n",nom_fichier);abort();}
    }

    //fermeture fichier
    int ret=fclose(fichier);
    if(ret!=0)
    {printf("Erreur ouverture fichier %s\n",nom_fichier);abort();}
    fichier=NULL;

    return 0;
}
```

# Ecriture/lecture fichier

## Exemple lecture fichier avec gestion d'erreur



```
int main()
{
    //base de donnees
    struct personne employe[4];

    //ouverture fichier
    char nom_fichier[]="ma_base.txt";
    FILE *fichier=NULL;
    fichier=fopen(nom_fichier,"r");
    if(fichier==NULL)
    {printf("Erreur ouverture fichier %s\n",nom_fichier);abort();}

    //ecriture fichier
    int k=0;
    for(k=0;k<4;++k)
    {
        int verif=fscanf(fichier,"%15s %15s %d %d \n",employe[k].nom,
            employe[k].fonction,
            &employe[k].salaire,
            &employe[k].anciennete);
        if(verif!=4)
        {printf("Erreur lecture fichier %s\n",nom_fichier);abort();}
    }

    //fermeture fichier
    int ret=fclose(fichier);
    if(ret!=0)
    {printf("Erreur fermeture fichier %s\n",nom_fichier);abort();}
    fichier=NULL;

    return 0;
}
```

# Bonnes pratiques Ecriture/Lecture

Toujours vérifier que **fopen** s'est bien déroulé



Toujours

Toujours

Toujours

- \* erreur nom
- \* chemin invalide  
(chemin locale dépend de l'endroit de l'execution!)
- \* fichier non existant
- \* fichier bloqué par une autre application

# Bonnes pratiques Ecriture/Lecture

Toujours vérifier que `fopen` s'est bien déroulé

Toujours

Toujours

Toujours

- \* erreur nom
- \* chemin invalide  
(chemin locale dépend de l'endroit de l'execution!)
- \* fichier non existant
- \* fichier bloqué par une autre application

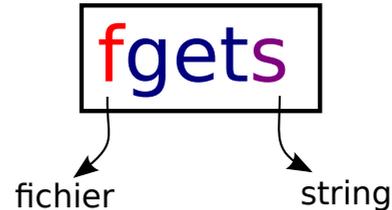
```
FILE *fid=NULL;  
fid=fopen(...);  
if(fid==NULL)  
{  
    ...  
}
```



# Bonnes pratiques Ecriture/Lecture

scanf("%s",...) => s'arrête au premier espace rencontré

↳ Pour lire une ligne complète:



`fgets(char buffer[], int taille_max, FILE *fichier)`

*lecture fichier*

```
#define TAILLE_MAX 128

int main()
{
    FILE *fichier=NULL;
    fichier=fopen("mon_fichier","r");

    //lit a partir d'un fichier
    char buffer_fichier[TAILLE_MAX];
    fgets(buffer_fichier, TAILLE_MAX, fichier);

    fclose(fichier);
}
```

*lecture clavier*

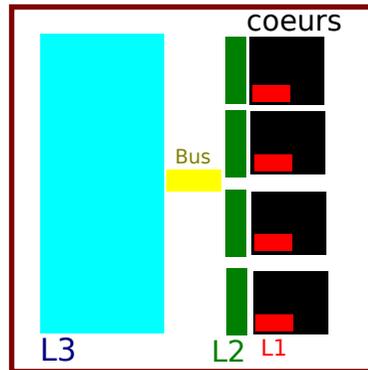
```
#define TAILLE_MAX 128

int main()
{
    //lit entree standard (clavier par default)
    char buffer_fichier[TAILLE_MAX];
    fgets(buffer_fichier, TAILLE_MAX, stdin);
}
```

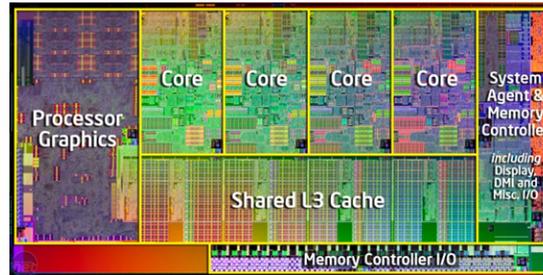
**rem.** Ne jamais utiliser ~~gets(...)~~  
=> Non sécurisé!

# Fichiers vitesse

	Transfert (Mo/s)	Tps Accès (ns)	Capacité (Mo)	Prix \$/Go
L1	60 000	0.5	0.032	on chip
L2	25 000	7	0.256	
L3	10 000	20	12	20 000 000
RAM	5 000	60	6000	20
Disque dur	150	10 000 000	1000000	0.1



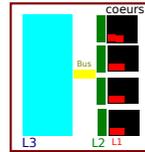
vue schématique  
Processeur récent



<http://www.ni.com>

# Fichiers vitesse

	Transfert (Mo/s)	Tps Accès (ns)	Capacité (Mo)	Prix \$/Go
L1	60 000	0.5	0.032	on chip
L2	25 000	7	0.256	
L3	10 000	20	12	20 000 000
RAM	5 000	60	6000	20
Disque dur	150	10 000 000	1000000	0.1



vue schématique  
Processeur récent

=> Ne pas abuser de lecture/écriture nombreuses opérations

→ Passer par un buffer, puis lire/écrire par blocs dans le fichier

=> Ne pas lire caractères par caractères dans un fichier

→ chargez l'ensemble dans un buffer  
puis lire dans le buffer

=> Ouvrir/fermer un même fichier trop souvent

→ Laisser le fichier ouvert tant que la fin de l'écriture n'a pas eu lieu

=> Ne pas utiliser de fichier pour des opérations critiques en temps

# Gestion des erreurs

Types d'erreurs

Erreur d'arrêt immédiats

Transmission d'information d'erreur

- retour d'arguments
- passage de paramètres

Erreur utilisateur

# Gestion des erreurs

Problème complexe

```
void ouverture_fichier(const char* filename)
{
    FILE* fid=NULL;
    fid=fopen(filename, "r");
    if(fid==NULL)
    {
        //GESTION D'ERREUR
    }
}
```

- Que faire?
- \* Quitter
  - \* Indiquer l'erreur
  - \* Revenir à la fonction appelante
  - \* Ecrire un log dans un fichier
  - \* Ne rien faire
  - \* Mettre une variable globale d'erreur à jour
  - \* ...

# Gestion des erreurs

```
void ouverture_fichier(const char* filename)
{
    FILE* fid=NULL;
    fid=fopen(filename, "r");
    if(fid==NULL)
    {
        //GESTION D'ERREUR
    }
}
```

**Constat:** plusieurs solutions possibles: dépend du contexte

ex. Si le fichier est censé exister (programmation par contrat)  
=> Erreur de programmation: il faut quitter et debugger

Si le fichier peut ne pas exister

=> Afficher ou non, et tenter potentiellement

 travail de la fonction appelante  
=> elle doit être mise au courant

# Glossaire des types d'erreurs

## 1. Erreur de programmation (/bugs)



ex.

*Non respect d'un contrat de programmation.*

*Indice de tableau <0 ou > taille*

*Ecriture sur un pointeur NULL, ou pointant sur une adresse non valide*

...

Doit être détectée le plus tôt possible

*rappel de la priorité*

```
+++ Compilateur
++ Par tests unitaires
+ Par assertions
- Par tests integrations
-- Par hasard par le programmeur
--- Par le client (catastrophe)
```



Que faire:

Aboutit à l'arrêt immédiat du programme  
Message d'erreur à l'attention d'un programmeur:  
Doit être suivi d'une modification du code

*doit être précis sur*  
*\* la cause*  
*\* endroit du code*

ex. `printf("Erreur du a ... ligne %d, fonction %s, fichier %s\n", __LINE__, __FUNCTION__, __FILE__);`

**Note:** Ne doit **Jamais** arriver en production/chez le client

# Glossaire des types d'erreurs

## 2. Erreur système critique



ex.

Manque de place mémoire RAM/disque  
(allocation mémoire qui échoue)

Operation importante non permise par le noyaux

Reception d'un signal de type "kill"

...

Généralement difficile à prévoir (dépend des conditions d'utilisations)

=> **Bien vérifier les retours des appels systèmes**

Que faire:

Aboutit généralement à l'arrêt du programme

(potentiellement après nettoyage: désallocation, suppression de fichiers temporaires, etc.)

Message d'erreur à l'attention de l'utilisateur (/potentiellement programmeur)

**Note:** Peut arriver chez le client

=> tests avec `if{...}` (et pas avec assertions)

(cas connu:

kernel panic / blue screen of death)

# Glossaire des types d'erreurs



## 3. Erreur système exceptionnelles non critique

ex.

Ressource occupée (fichier, variable partagée, etc.)

Fichier de configuration manquant ou invalide

...

Note: L'aspect "exceptionnelle" et "critique" dépend de l'utilisation

Que faire:

Aboutit au traitement spécifique par la fonction appelante.

(temporisation, création de nouveaux fichiers, utilisation de valeurs par défauts, ...)

Message d'erreur à l'attention de l'utilisateur en mode debug

(pas forcément d'alerte en mode release si correctement traité).

```
ex. char fichier[]="mon_fichier_de_configuration.xml";
if(valide_fichier_configuration(fichier)==0)
{
    if(mode_debug==1)
        printf("Attention, fichier configuration %s non valide\n",fichier);
    initialise_valeur_par_defaut(fichier);
}
```

**Note:** Va sans doute arriver chez le client

=> tests avec if{...} (et pas avec assertions)

Utilisation de **exception**

(voir Java, C++, ...)

# Glossaire des types d'erreurs

## 4. Erreur utilisateur (Ce n'est pas une erreur)



ex.  
URL inexistante  
Identifiant non connu dans la base  
Demande d'un nombre positif, et recoit -12  
...

Que faire:

Ne surtout pas quitter brutalement le programme (SegFault, abort) !  
Message d'information à l'attention de l'utilisateur uniquement.  
Doit être traité par une fonction spécifique de validation d'entrée utilisateur.

**Note:** Arrive forcément chez le client

=> tests avec `if{...}` (et pas avec assertions)

# Gestion des erreurs

Types d'erreurs

→ **Erreur d'arrêt immédiats**

Transmission d'information d'erreur

- retour d'arguments
- passage de paramètres

Erreur utilisateur

# Erreur d'arrêt immédiat

## Approche manuelle

```
#define AFFICHE_INFO_DEBUG printf("l.%d, %s, %s\n",__LINE__,__FUNCTION__,__FILE__)\n\n//Contrat: recoit entier a de valeur absolue plus petite que 1000\nvoid ma_fonction(int a)\n{\n    int contrat_valide=fabs(a<1000);\n    if(contrat_valide==0)\n    {\n        //affiche le type d'erreur\n        printf("Erreur a [%d] doit etre plus petit que 1000\n",a);\n        //affiche les informations de localisation pour le debug\n        AFFICHE_INFO_DEBUG;\n        //quitte brutalement\n        exit(1);\n    }\n}
```

renvoi 1 à la ligne de commande

ex. ma\_fonction(10000) => Erreur a [10000] doit etre plus petit que 1000  
l.13, ma\_fonction, mon\_fichier.c

# Erreur d'arrêt immédiat

Approche par assertion



```
#include <assert.h>

//Contrat: recoit entier a de valeur absolue plus petite que 1000
void ma_fonction(int a)
{
    assert(a<1000);
}
```

ex. ma\_fonction(10000) =>

```
mon_executable: mon_fichier.c:8: ma_fonction: Assertion `a<1000' failed.
Aborted
```

# Erreur d'arrêt immédiat

## Comparaison if/assert

```
if(certificat==0)
{
    printf(<type d'erreur>+<info_debug>);
    exit(<valeur_erreur>) / abort();
}
```

- + de liberté:
- message erreur
- la manière de quitter

```
assert(certificat);
```

- + cours
- => **+lisible**
- message erreur complet pour debug
- => **pas de risques d'oublis**
- standard
- => **auto-documentation**

Conclusion 1: Pour des erreurs de **developpement**  
=> Utilisez **assert!**



# Erreur d'arrêt immédiat



Etude du fonctionnement de assert:

```
#include <assert.h>

//Contrat: recoit entier a de valeur absolue plus petite que 1000
void ma_fonction(int a)
{
    assert(a<1000);
}
```

*compilation mode debug*

`gcc -g mon_fichier.c -E  
-Wall -Wextra`

```
void ma_fonction(int a)
{
    ((a<1000) ? (void) (0) : __assert_fail ("a<1000", "mon_fichier.c", 7, __PRETTY_FUNCTION__));
}
```

-> utilisation des macros à votre place

# Erreur d'arrêt immédiat



Etude du fonctionnement de assert:

```
#include <assert.h>

//Contrat: recoit entier a de valeur absolue plus petite que 1000
void ma_fonction(int a)
{
    assert(a<1000);
}
```

*compilation mode release*

```
gcc -O2 mon_fichier.c -E
-Wall -Wextra
-DNDEBUG
```

```
void ma_fonction(int a)
{
    ((void) (0));
}
```

-> condition non vérifiée en mode *release*

# Erreur d'arrêt immédiat

Etude du fonctionnement de assert:



**Synthèse:** Assert() ne vérifie la condition qu'en mode **debug**  
=> pendant le developpement

## Avantage:

Pas de perte de temps en release

## Inconvénient:

Ne peut être utilisé (seul) que pour des erreurs de developpement n'arrivant jamais sur la version finale

**Conclusion:** Assert() est une **aide pour le developpeur!**  
Ce n'est pas une vérification constante

# Erreur d'arrêt immédiat

Exemple de assert:

```
//Contrat: recoit un pointeur non NULL
//Certification: la valeur du pointeur vaut 2
void ecrit_deux(int *pointeur)
{
    assert(pointeur!=NULL);
    *pointeur=2;
}

int main()
{
    int a;
    ecrit_deux(&a);
    ecrit_deux(NULL);
}
```

Mode debug

`gcc -g -Wall -Wextra`

mon\_executable: erreur\_01.c:8: ecrit\_deux: Assertion `pointeur!=((void \*)0)' failed.  
Aborted

Mode release

`gcc -O2 -Wall -Wextra -DNDEBUG`

Segmentation fault

# Erreur d'arrêt immédiat

Exemple de assert:

```
void demande_valeur_utilisateur()  
{  
    int nombre=0;  
  
    printf("Donnez nombre entre 0 et 10.\n> ");  
    scanf("%d",&nombre);  
  
    assert(nombre>=0 && nombre<=10);  
  
    printf("Vous avez choisi le nombre %d\n",nombre);  
}
```

A ne pas faire!

X

=> Ne pas vérifier une entrée utilisateur avec un assert

- > quitte brutalement en mode debug = mauvais
- > ne vérifie rien en mode release = mauvais

# Erreur d'arrêt immédiat

Manipulation de tableau sans dépassement mémoire:

```
#define MAX_INDICE 4
struct vecteur
{
    int tableau[MAX_INDICE];
};

//Pre-requis:
// Pointeur constant vers un vecteur. Pointeur non NULL.
// Indice pointant dans ce vecteur: entier non signe < MAX_INDICE
//Certifie:
// Renvoie la valeur du vecteur pointe par cet indice
int vecteur_get(const struct vecteur* v,unsigned int indice)
{
    //certification de developpement
    assert(v!=NULL);
    assert(indice<MAX_INDICE);

    return v->tableau[indice];
}

int main()
{
    struct vecteur v={ {1,2,4,5} };

    int a=vecteur_get(&v,2);
    int b=vecteur_get(&v,5);

    return 0;
}
```

## Avantages:

*Lors du développement:*

Assure le non dépassement mémoire

a.out: erreur\_01.c:20: vecteur\_get: Assertion `indice<4' failed.

*Lors de l'utilisation:*

Aucune perte de performance par rapport à la version sans verification.

# Erreur d'arrêt immédiat



Cumule aide au developpement / bugs a l'utilisation

```
//Pre-requis:
// Pointeur constant vers un vecteur. Pointeur non NULL.
// Indice pointant dans ce vecteur: entier non signe < MAX_INDICE
//Certifie:
// Renvoie la valeur du vecteur pointe par cet indice
int vecteur_get(const struct vecteur* v, unsigned int indice)
{
    //certification de developpement
    assert(v!=NULL);
    assert(indice<MAX_INDICE);
    //mauvaise utilisation en production
    if(indice>=MAX_INDICE)
    {
        printf("Attention evenement inattendu survenue\n");
        printf("Contactez le developpeur avec les informations du fichier log.txt\n");

        //ecriture des informations de debugs dans log.txt
        FILE *fid=NULL;
        fid=fopen("log.txt", "w");

        if(fid==NULL)//si on cumule les problemes
            {printf("Erreur ecriture fichier de log.txt, je quitte\n");abort();}

        fprintf(fid, "Erreur depassement indice %d>%d\n", indice, MAX_INDICE);
        fprintf(fid, "l.%d; %s; %s\n", __LINE__, __FUNCTION__, __FILE__);

        fclose(fid);
        return -1;//retourne une valeur par default => evite la seg-fault
    }
    return v->tableau[indice];
}
```

gcc -O2 -Wall -Wextra -DNDEBUG

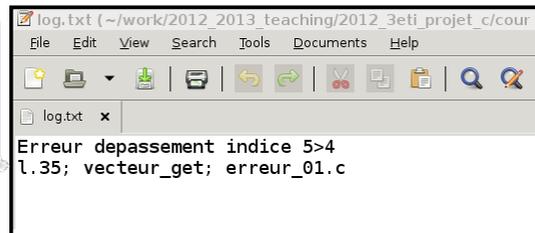
Attention evenement inattendu survenue  
Contactez le developpeur avec les informations du fichier log.txt

+ le programme ne s'arrête pas

+



log.txt



# Gestion des erreurs

Types d'erreurs

Erreur d'arrêt immédiats

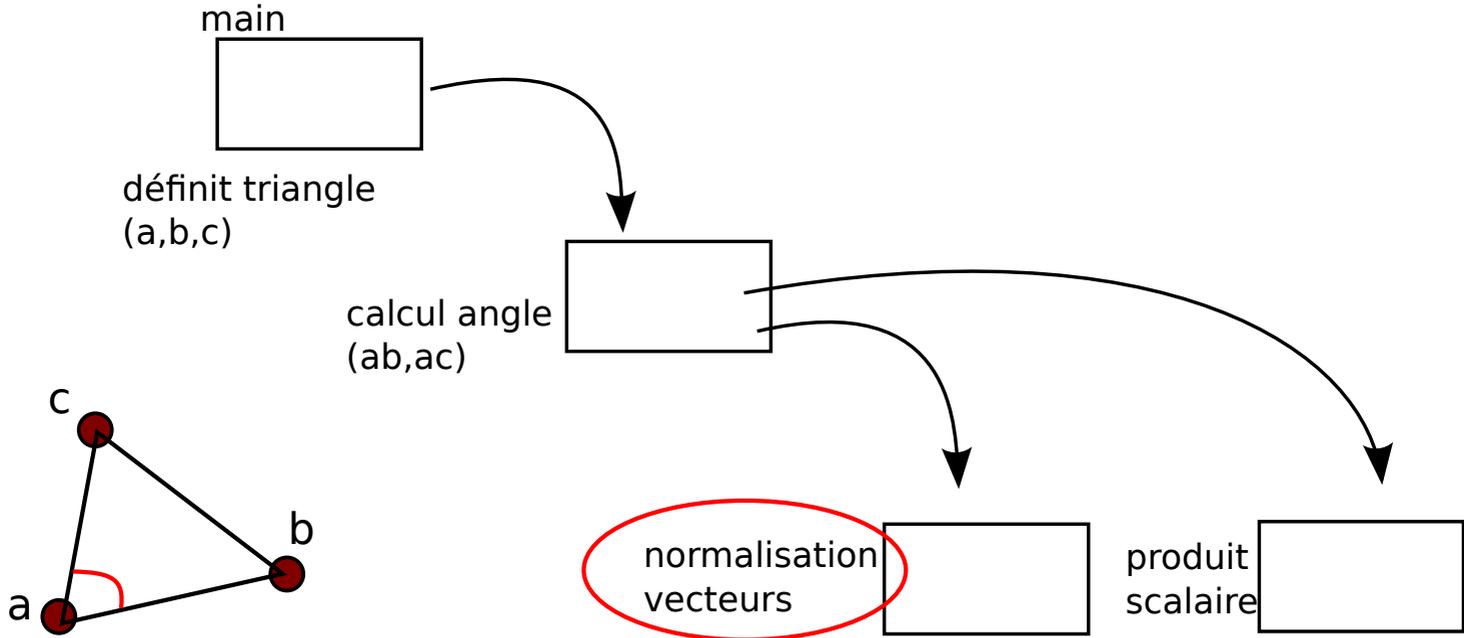
→ **Transmission d'information d'erreur**

- retour d'arguments
- passage de paramètres

Erreur utilisateur

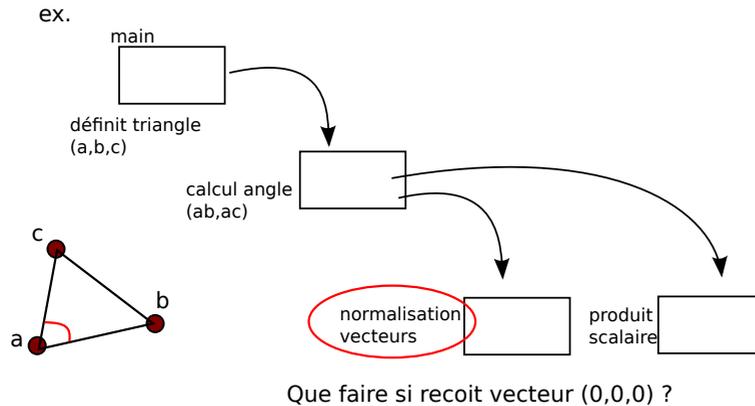
# Erreur gérée par la fonction appelante

ex.



Que faire si recoit vecteur (0,0,0) ?

# Erreur gérée par la fonction appelante



Dans normalisation vecteurs (localement), on ne peut que:

- \* Quitter => acceptable ?
- \* Renvoyer une valeur par défaut => laquelle?

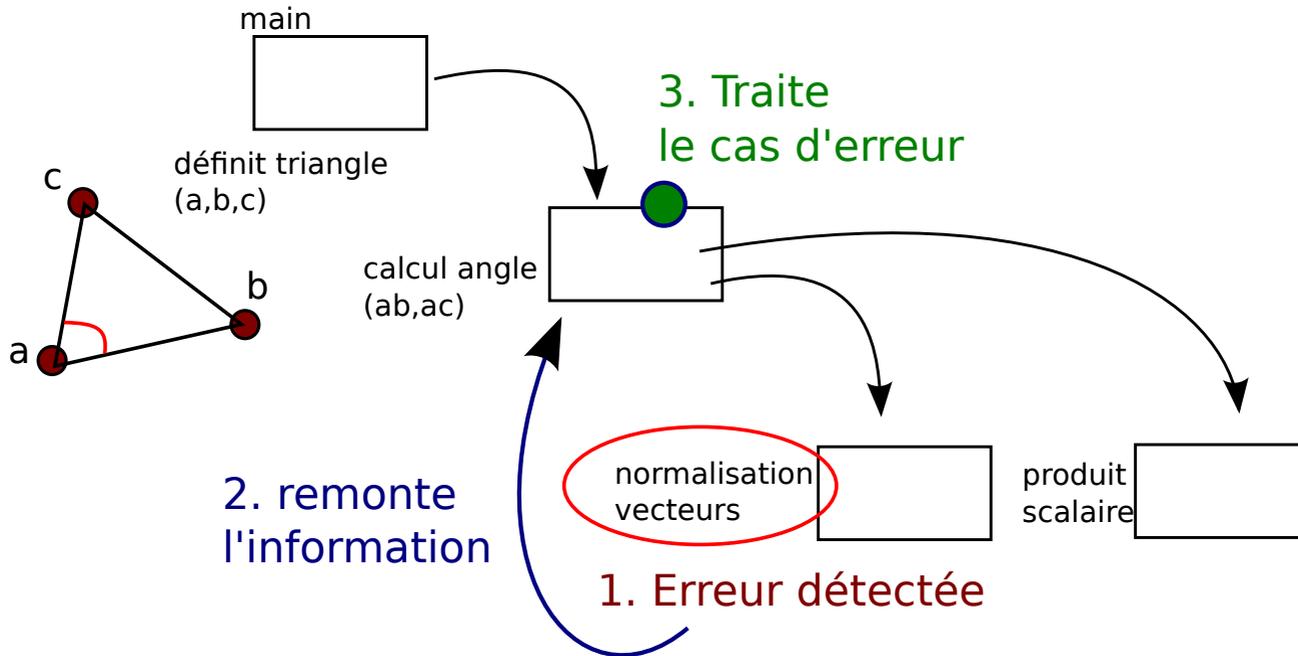
Mieux:

Indiquer à *calcul\_angle*, ou dans le *main* le problème

→ renvoi un angle de 0 par exemple

# Erreur gérée par la fonction appelante

Il faut faire remonter l'information d'une erreur



# Erreur gérée par la fonction appelante

Plusieurs solutions possible:



- \* Valeur de retour
- \* Variable globales
- \* Passage d'arguments par pointeurs

) un mix de tous

Hors C

- \* Exception

But:

- Garder un programme lisible!
- Garder un programme maintenable
- Rester transparent en cas de bon fonctionnement  
(sinon perte de lisibilité)
- Pas trop transparent  
(sinon oubli de gestion des cas d'erreurs)

# Erreur par retour d'arguments

```
fonction:
#define FAIL 0
#define OK 1

int ecrit_fichier(const char *filename)
{
    assert(filename!=NULL);

    FILE *fid=NULL;
    fid=fopen(filename, "w");

    if(fid==NULL)
        return FAIL; //erreur

    int nbr_lettre=fopen(fid, "Coucou c'est moi\n");
    if(nbr_lettre!=18)
        return FAIL; //erreur

    int ret=fclose(fid);
    if(ret!=0)
        return FAIL; //erreur

    return OK;
}
```

## Avantage:

- utilisation assez élégante  
=> if(ma\_fonction())
- potentiellement transparent  
=> possibilité de non vérification

## Inconvénient:

- Empêche le retour d'arguments

```
utilisation:
int main()
{
    if(ecrit_fichier("mon_fichier.txt")==FAIL)
    {
        printf("mon_fichier.txt n'est pas accessible, tente d'ecrire sur mon_fichier_2.txt\n");

        if(ecrit_fichier("mon_fichier_2.txt")==FAIL)
        {
            printf("Ce n'est pas mon jour de chance, je fais autre chose");
            faire_autre_chose();
        }
    }

    return 0;
}
```

# Erreur par retour d'arguments

```
#define FAIL 0
#define OK 1

struct vecteur
{float x,y,z;};

//Prend en parametre un vecteur et retourne le vecteur unitaire de meme direction
int vecteur_unitaire(struct vecteur vec,struct vecteur* resultat)
{
  assert(resultat!=NULL);

  float norme=sqrt(vec.x*vec.x+vec.y*vec.y+vec.z*vec.z);

  if(norme<1e-5)
    return FAIL;

  resultat->x = vec.x/norme;
  resultat->y = vec.y/norme;
  resultat->z = vec.z/norme;

  return OK;
}
```

```
int main()
{
  struct vecteur a={1,2,3};struct vecteur ua={1,1,1};
  struct vecteur b={0,0,0};struct vecteur ub={1,1,1};

  int ret_1=vecteur_unitaire(a,&ua);
  int ret_2=vecteur_unitaire(b,&ub);

  if(ret_1!=OK || ret_2!=OK)
  {
    printf("Attention, vecteur de norme nulle\n");
    printf("Angle = 0.0\n");
    return 0;
  }

  printf("Angle = %f\n",ua.x*ub.x+ua.y*ub.y+ua.z*ub.z);

  return 0;
}
```

peu lisible, pas pratique.

on préfèrerait:  
`struct vecteur ua=vecteur_unitaire(a);`

# Erreur par retour d'arguments

```
struct vecteur
{float x,y,z;};

float vecteur_norme(struct vecteur vec)
{
    return sqrt(vec.x*vec.x+vec.y*vec.y+vec.z*vec.z);
}

int vecteur_est_norme_non_nulle(struct vecteur vec)
{
    float epsilon=1e-5;
    return vecteur_norme(vec)>epsilon;
}

//Prend en parametre un vecteur et retourne le vecteur unitaire de meme direction
// Prerequis: Un vecteur de norme non nulle
// Certifie: Un vecteur de meme direction et de norme unitaire
struct vecteur vecteur_unitaire(struct vecteur vec)
{
    assert(vecteur_est_norme_non_nulle(vec)==VRAI);

    float norme=vecteur_norme(vec);
    struct vecteur resultat={vec.x/norme, vec.y/norme, vec.z/norme};

    return resultat;
}
```

fonction "binaire" de  
vérification amont

utilisation de contrat pour éviter les  
erreurs de codage

```
int main()
{
    struct vecteur a={1,2,3}; struct vecteur ua={1,1,1};
    struct vecteur b={0,0,0}; struct vecteur ub={1,1,1};

    if(vecteur_est_norme_non_nulle(a)==FAUX || vecteur_est_norme_non_nulle(b)==FAUX)
    {
        printf("Attention, vecteur de norme nulle\n");
        printf("Angle = 0.0\n");
        return 0;
    }

    ua=vecteur_unitaire(a);
    ub=vecteur_unitaire(b);

    printf("Angle = %f\n", ua.x*ub.x+ua.y*ub.y+ua.z*ub.z);

    return 0;
}
```

+ lisible  
+ logique

traitement spécifique  
erreur "utilisateur"

# Erreur par retour d'arguments

Le type d'erreur peut être mixé avec le résultat **A éviter!**

```
#define TAILLE_TABLEAU 6

int recupere_note(const int* tableau_note,int indice)
{
    if(tableau_note==NULL)
        return -1;
    if(indice<0)
        return -2;
    if(indice>TAILLE_TABLEAU)
        return -3;

    return tableau_note[indice];
}

int main()
{
    //les notes sont comprises entre 0 et 20
    int tableau_note[TAILLE_TABLEAU]={12,14,5,14,16,18};

    int valeur_erreur=recupere_note(tableau_note,12);
    if(valeur_erreur<0)
    {
        switch(valeur_erreur)
        {
            case -1:
                printf("pointeur NULL\n");break;
            case -2:
                printf("indice negatif\n");break;
            case -3:
                printf("indice trop grand\n");break;
        }
    }
    else
        printf("Note %d\n",valeur_erreur);
}
```

## Avantage:

Compact (si peu de mémoire)

Effet de bords

(si le domaine de validité des données changent)

## Inconvénient:

Perte de lisibilité

Mélange données/erreurs

Documentation difficile

**=> A éviter**

# Erreur par retour d'arguments

mélange code erreur / valeurs par mask de bits



```
//couleur:  
//0xEERRVVBB  
//EE: erreur -> 01: indice trop petit  
//   erreur -> 02: indice trop grand  
//RR: rouge  
//VV: vert  
//BB: bleu
```

```
#define TAILLE_TABLEAU 4
```

```
int recupere_couleur(int tableau_couleur[],int indice)  
{  
    if(indice<0)  
        return 0x01000000;  
    if(indice>TAILLE_TABLEAU)  
        return 0x02000000;  
  
    return tableau_couleur[indice];  
}
```

Concaténation (code\_erreur,rouge,vert,bleu)

## A NE PAS FAIRE!

Inconvénients:

- difficile à lire
- documentation indispensable
- pas évolutif  
(ajout canal alpha impossible)

```
int main()  
{  
    //les notes sont comprises entre 0 et 20  
    int tableau_couleur[TAILLE_TABLEAU]={0x000000,0xFF0000,0x00FF00,0x0000FF};  
  
    int valeur_erreur=recupere_couleur(tableau_couleur,8);  
    if(valeur_erreur>>24)  
    {  
        switch(valeur_erreur>>24)  
        {  
            case 1:  
                printf("indice negatif\n");break;  
            case 2:  
                printf("indice trop grand\n");break;  
        }  
    }  
    else  
        printf("Couleur (%x,%x,%x)\n",  
            valeur_erreur>>16,  
            (valeur_erreur&0x00FFFF)>>8,  
            (valeur_erreur&0x0000FF));  
}
```



=> Ne pas utiliser ce genre d'approche pour du developpement d'un logiciel.

OK uniquement pour embarqué limité

peu lisible

# Erreur par passage de paramètre

Cas d'erreur écrit dans un paramètre passe par pointeur  
(si pointeur NULL => pas de prise en compte)

```
#define FAUX 0
#define VRAI 1

struct vecteur
{float x,y,z;};

float vecteur_norme(struct vecteur vec)
{
    return sqrt(vec.x*vec.x+vec.y*vec.y+vec.z*vec.z);
}

struct vecteur vecteur_unitaire(struct vecteur vec,int *execution_ok)
{
    struct vecteur resultat={1,0,0};
    float norme=vecteur_norme(vec);
    if(norme<1e-5)
    {
        if(execution_ok!=NULL)
            *execution_ok=FAUX;
        return resultat;
    }
    resultat.x=vec.x/norme;
    resultat.y=vec.y/norme;
    resultat.z=vec.z/norme;

    if(execution_ok!=NULL)
        *execution_ok=VRAI;

    return resultat;
}
```

```
int main()
{
    struct vecteur a={0,0,0};

    int execution_ok=VRAI;
    struct vecteur ua=vecteur_unitaire(a,&execution_ok);

    if(execution_ok==FAUX)
    {
        printf("Erreur de normalisation de vecteur\n");
        return 0;
    }

    struct vecteur ub=vecteur_unitaire(a,NULL);

    return 0;
}
```

prise en compte de l'erreur

on ne tiens pas compte du cas d'erreur

# Erreur par passage de paramètre

Cas d'erreur écrit dans un paramètre passe par pointeur  
(si pointeur NULL => pas de prise en compte)

```
#define FAUX 0
#define VRAI 1

struct vecteur
{float x,y,z;};

float vecteur_norme(struct vecteur vec)
{ return sqrt(vec.x*vec.x+vec.y*vec.y+vec.z*vec.z);
}

struct vecteur vecteur_unitaire(struct vecteur vec,int *execution_ok)
{
  struct vecteur resultat=(1,0,0);
  float norme=vecteur_norme(vec);
  if(norme<1e-5)
  {
    if(execution_ok!=NULL)
      *execution_ok=FAUX;
    return resultat;
  }
  resultat.x=vec.x/norme;
  resultat.y=vec.y/norme;
  resultat.z=vec.z/norme;
  if(execution_ok!=NULL)
    *execution_ok=VRAI;
  return resultat;
}

int main()
{
  struct vecteur a={0,0,0};
  int execution_ok=VRAI;
  struct vecteur ua=vecteur_unitaire(a,&execution_ok);

  if(execution_ok==FAUX)
  {
    printf("Erreur de normalisation de vecteur\n");
    return 0;
  }

  struct vecteur ub=vecteur_unitaire(a,NULL);

  return 0;
}
```

prise en compte de l'erreur

on ne tiens pas compte du cas d'erreur

## Avantage:

- \* permet l'écriture `a=fonction(b,...)`  
=> utilisation plus lisible
- \* possibilité de ne pas tenir compte de l'erreur  
mais reste visible

## Inconvénient:

- \* documentation nécessaire  
(paramètre NULL?)
- \* syntaxe fonction moins lisible

# Erreur par passage de paramètre

Possibilité de travailler avec une struct  
(extensibilité, lisibilité)

```
//les différents types d'erreur possibles
enum type_erreur {pointeur_null, indice_trop_grand, indice_trop_petit};

#define TAILLE_MAX 256
struct code_erreur
{
    int actif; //est ce qu'une erreur existe

    enum type_erreur type; //le type d'erreur
    int ligne; //la ligne d'ou provient l'erreur
    char nom_fonction[TAILLE_MAX]; //le nom de la fonction
                                    //d'ou provient l'erreur
};
```

structure d'erreur  
+ contient toutes les infos de debug  
+ lisible  
=> extensible

```
int main()
{
    int tableau[TAILLE_TABLEAU]={4, 2, -1, 4};

    int a=recupere_valeur(tableau, 2, NULL);

    struct code_erreur erreur={ FAUX, 0, 0, "\0" };
    int b=recupere_valeur(tableau, -3, &erreur);

    if(erreur.actif==VRAI)
        traitement_erreur(&erreur);

    return 0;
}
```

utilisation  
transparente

gestion du cas d'erreur

# Erreur par passage de paramètre

Possibilité de travailler avec une struct  
(extensibilité, lisibilité)

```
#define VRAI 1
#define FAUX 0

#define TAILLE_TABLEAU 4

int recupere_valeur(const int* tableau,int indice,struct code_err
eur *retour_erreur)
{
    if(tableau==NULL)
    {
        if(retour_erreur!=NULL)
        {
            retour_erreur->actif=VRAI;
            retour_erreur->type=pointeur_null;
            retour_erreur->ligne=__LINE__;
            strcpy(retour_erreur->nom_fonction,__FUNCTION__);
        }
        return -1;
    }

    if(indice<0)
    {
        if(retour_erreur!=NULL)
        {
            retour_erreur->actif=VRAI;
            retour_erreur->type=indice_trop_petit;
            retour_erreur->ligne=__LINE__;
            strcpy(retour_erreur->nom_fonction,__FUNCTION__);
        }
        return -1;
    }

    if(indice>=TAILLE_TABLEAU)
    {
        if(retour_erreur!=NULL)
        {
            retour_erreur->actif=VRAI;
            retour_erreur->type=indice_trop_grand;
            retour_erreur->ligne=__LINE__;
            strcpy(retour_erreur->nom_fonction,__FUNCTION__);
        }
        return -1;
    }

    if(retour_erreur!=NULL)
        retour_erreur->actif=FAUX;

    return tableau[indice];
}
```

```
//les differents types d'erreur possibles
enum type_erreur {pointeur_null,indice_trop_grand,indice_trop_petit};

#define TAILLE_MAX 256
struct code_erreur
{
    int actif; //est ce qu'une erreur existe

    enum type_erreur type; //le type d'erreur
    int ligne; //la ligne d'ou provient l'erreur
    char nom_fonction[TAILLE_MAX]; //le nom de la fonction
                                //d'ou provient l'erreur
};
```

une implémentation possible  
de la fonction, et du traitement de l'erreur:

```
void traitement_erreur(const struct code_erreur* e)
{
    assert(e->actif==VRAI);

    printf("Erreur detecte\n");
    printf("Erreur de type %d\n",e->type);
    printf("a la ligne %d, dans la fonction %s\n",
        e->ligne,e->nom_fonction);
}
```

# Erreur par passage de paramètre

## Communication par variable globale

```
//constantes de gestion d'erreurs
#define PAS_ERREUR 0
#define ERREUR_DIVISION_ZERO 1

//variable globale d'erreur
int variable_globale_erreur=PAS_ERREUR;

struct vecteur
{float x,y,z;};

struct vecteur vecteur_unitaire(struct vecteur vec)
{
    float norme=vec.x*vec.x+vec.y*vec.y+vec.z*vec.z;
    float epsilon=1e-5;

    struct vecteur resultat={-1,-1,-1};
    if(norme<epsilon)
    {
        variable_globale_erreur=ERREUR_DIVISION_ZERO;
        return resultat;
    }

    resultat.x=vec.x/norme;
    resultat.y=vec.y/norme;
    resultat.z=vec.z/norme;

    return resultat;
}
```

```
int main()
{
    struct vecteur a={1,2,3};
    struct vecteur b={0,0,0};

    struct vecteur ua=vecteur_unitaire(a);
    struct vecteur ub=vecteur_unitaire(b);

    if(variable_globale_erreur!=PAS_ERREUR)
        {printf("Attention, vecteur de norme nulle detectee \n");return 1;}

    return 0;
}
```

### Avantage:

- transparent
- approche standard C

### Inconvénient:

- trop transparent  
=> absence de vérification
- documentation nécessaire

# Gestion des erreurs

Types d'erreurs

Erreur d'arrêt immédiats

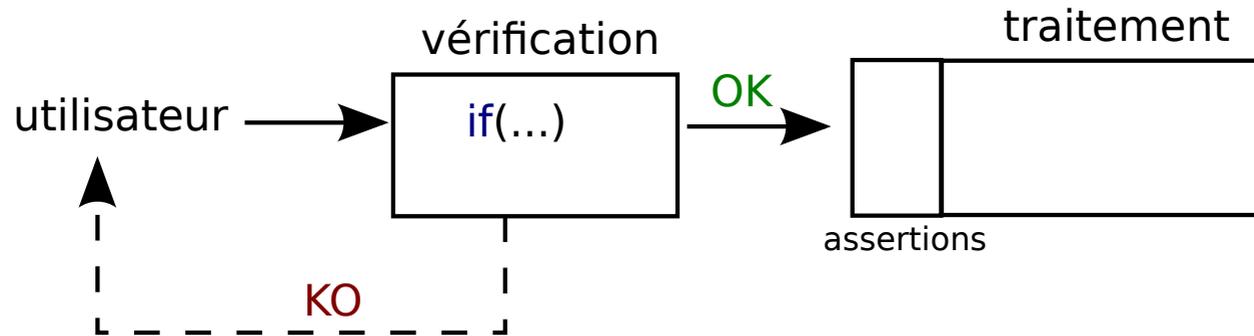
Transmission d'information d'erreur

- retour d'arguments
- passage de paramètres

→ **Erreur utilisateur**

# Erreur utilisateur

Séparer vérification du traitement



**But:**

- Construire des blocs unitaires (saisie, traitement)
- Chaque bloc est certifié
- Chaque bloc possède la gestion d'erreur appropriée

# Erreur utilisateur

ex.

Programme affichant un nom de fichier existant donné par l'utilisateur

```
#define TAILLE_MAX 128
#define TAILLE_LIGNE 256

int main()
{
    char filename[TAILLE_MAX];

    int fichier_ok=0;
    do
    {
        //saisie
        printf("Indiquez chemin vers fichier:\n> ");
        scanf("%128s", filename);

        //test existence
        FILE *fid=NULL;
        fid=fopen(filename, "r");

        if(fid!=NULL)
        {
            fichier_ok=1;

            //lecture et affichage du fichier
            char buffer[TAILLE_LIGNE];
            while (fgets(buffer, TAILLE_LIGNE, fid) != NULL)
                printf("%s", buffer);

            fclose(fid);
            fid=NULL;
        }
        else
            printf("Fichier %s n'existe pas\n\n", filename);
    }while(fichier_ok!=1);

    return 0;
}
```

**A EVITER!**

**Désavantage:**

Mélange utilisateur/traitement  
=> difficile à comprendre/lire

traitement non unitaire  
=> difficile à debugger/tester

Traitement et saisie non réutilisable  
dans un autre contexte

partie utilisateur

partie traitement

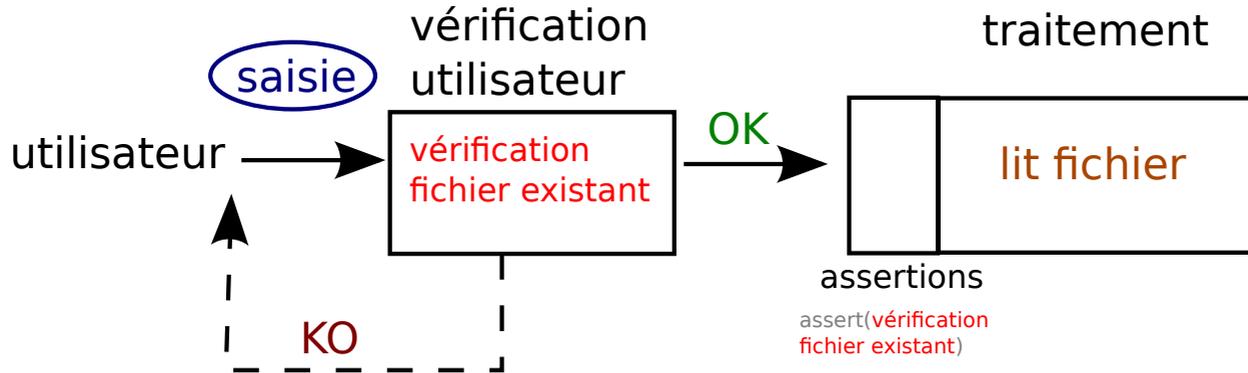
partie utilisateur

# Erreur utilisateur

ex.

Programme affichant un nom de fichier existant donné par l'utilisateur

Amélioration possible:



**saisie** = demander nom de fichier en ligne de commande

**vérification fichier existant** = ouvrir fichier, vérifier que l'ouverture est correcte

**lit fichier** = ouvre fichier, lit ligne après ligne jusqu'à fin, ferme fichier.

# Erreur utilisateur

ex.

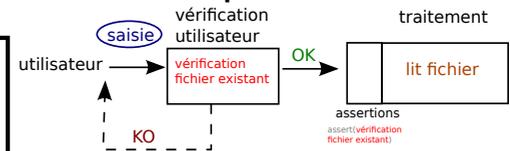
Programme affichant un nom de fichier existant donné par l'utilisateur

```
//Donnez le nom d'un fichier a lire
#define TAILLE_MAX 128
#define TAILLE_LIGNE 256

//Saisie utilisateur d'un fichier
//Demande a l'utilisateur de saisir le chemin vers
// un fichier existant.
void saisie_fichier_existant(char filename[]);

//Verifie si un fichier est accessible en mode lecture
//Prerequis:
//    Un nom de fichier sous forme de chaine de caracteres.
//Certifie:
//    Retourne 0 si le fichier n'est pas accessible en mode lecture
//    Retourne 1 si le fichier est accessible en mode lecture
int est_fichier_existant(const char filename[]);

//Lecture d'un fichier existant
//Prerequis:
//    Un nom de fichier qui existe deja.
//Certifie:
//    L'affichage sur la ligne de commande du fichier.
void lecture_fichier(char filename[]);
```



```
int main()
{
    char filename[TAILLE_MAX]="\0";

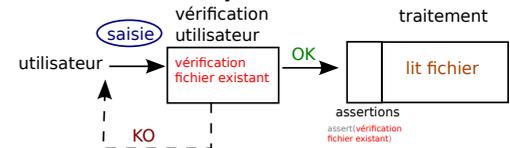
    saisie_fichier_existant(filename); //bloc saisie utilisateur
    lecture_fichier(filename);        //bloc traitement

    return 0;
}
```

# Erreur utilisateur

ex.

Programme affichant un nom de fichier existant donné par l'utilisateur



Implémentation possible:

```
int est_fichier_existant(const char filename[])
{
    FILE *fid=NULL;
    fid=fopen(filename, "r");
    if(fid==NULL)
        return 0;

    int ret=fclose(fid);
    if(ret!=0){printf("Probleme fermeture fichier %s\n", filename);}
    fid=NULL;
    return 1;
}
```

```
void lecture_fichier(char filename[])
{
    assert(est_fichier_existant(filename));

    FILE *fid=NULL;
    fid=fopen(filename, "r");

    assert(fid!=NULL);

    //lecture et affichage du fichier
    char buffer[TAILLE_LIGNE];
    while(fgets(buffer, TAILLE_LIGNE, fid) != NULL)
        printf("%s", buffer);

    int ret=fclose(fid);
    assert(ret==0);
    fid=NULL;
}
```

```
void saisie_fichier_existant(char filename[])
{
    int fichier_ok=0;

    do
    {
        //saisie
        printf("Indiquez chemin vers fichier:\n> ");
        scanf("%128s", filename);

        //test existence
        fichier_ok=est_fichier_existant(filename);

        if(fichier_ok!=1)
            printf("Fichier %s n'existe pas\n\n", filename);
    }while(fichier_ok!=1);
}
```

# Méthode de debug

Méthode manuelle (printf)  
Debugger à points d'arrêts  
Détecteur fuites mémoires

# Methode de debug

*Bugs* de:

Fonctionnement globale

=> Principe de tests: unitaire + intégrations

Erreurs mémoires

=> utilisation de debuggers

Segmentation fault = erreur mémoire  
Erreur de segmentation

= écriture/accès à une zone mémoire non autorisée

(C'est le noyau qui lance une segfault: pas le programme!)

90% du temps:

*Segmentation fault* due à :

- \* dépassement de tableau (indice trop grand ou  $<0$ )
- \* écriture sur pointeur non alloué
- \* désallocation sur une adresse invalide (free)

# Methode de debug

Difficile à débbugger car:

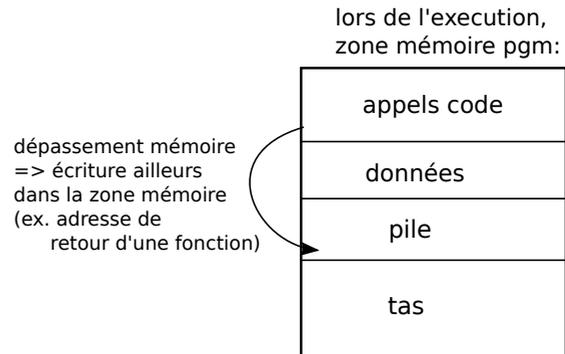
- \* Segfault ou comportement inattendu **non détecté** au moment de l'erreur
- \* Une erreur mémoire n'aboutit **pas forcément** à une segmentation fault



- \* peut aboutir à un comportement indéterminé (aléatoire en fonction des exécutions)
- \* peut ne pas apparaître
- \* peut modifier le comportement hors de la logique du C
- \* le comportement suspect n'apparaît pas au moment de l'erreur, mais plus loin dans le programme

ex typiques:

- \* Valeur d'un tableau ou d'une variable modifiée
- \* Comportement du programme différent lors de l'ajout de commentaires dans le code
- \* Le return d'une fonction n'est pas récupéré par la fonction appelante



# Méthode de debug

→ **Méthode manuelle (printf)**

Debugger à points d'arrêts

Détecteur fuites mémoires

# Debug par affichage: printf

```
#define TAILLE_MAX 15

void derivation(int valeurs[],int derivee[])
{
    int k=0;
    for(k=0;k<TAILLE_MAX;++k)
        derivee[k] = valeurs[k+1]-valeurs[k];
}

int main()
{
    int valeurs[TAILLE_MAX];
    int derivee[TAILLE_MAX];

    int k=0;
    for(k=0;k<TAILLE_MAX;++k)
        valeurs[k]=3*k*k;

    derivation(valeurs,derivee);

    return 0;
}
```

```
#define TAILLE_MAX 15

void derivation(int valeurs[],int derivee[])
{
    int k=0;
    for(k=0;k<TAILLE_MAX;++k)
    {
        printf("%d/%d -> %d\n",k,TAILLE_MAX,valeurs[k]);
        printf("%d/%d -> %d\n",k+1,TAILLE_MAX,valeurs[k+1]);
        derivee[k] = valeurs[k+1]-valeurs[k];
    }
}

int main()
{
    int valeurs[TAILLE_MAX];
    int derivee[TAILLE_MAX];

    int k=0;
    for(k=0;k<TAILLE_MAX;++k)
        valeurs[k]=3*k*k;

    derivation(valeurs,derivee);

    return 0;
}
```

```
0/15 -> 0
1/15 -> 3
1/15 -> 3
2/15 -> 12
2/15 -> 12
...
13/15 -> 507
13/15 -> 507
14/15 -> 588
14/15 -> 588
15/15 -> 15
```

# Debug par affichage: printf

Comportement inattendu:

```
int main()
{
    char mot_1[]="coucou";
    char mot_2[]="bonjour monsieur";

    strcpy(mot_2," en fin de journee, nous disons bonsoir");
    printf("%s\n",mot_1);

    return 0;
}
```

```
./mon_executable
> bonsoir
```

```
int main()
{
    char mot_1[]="coucou";
    char mot_2[]="bonjour monsieur";

    printf("%d/%d\n",sizeof(mot_2)/sizeof(char),
           strlen(" en fin de journee, nous disons bonsoir")+1);

    strcpy(mot_2," en fin de journee, nous disons bonsoir");

    printf("%s\n",mot_1);

    return 0;
}
```

```
./mon_executable
> 17/40
> bonsoir
```

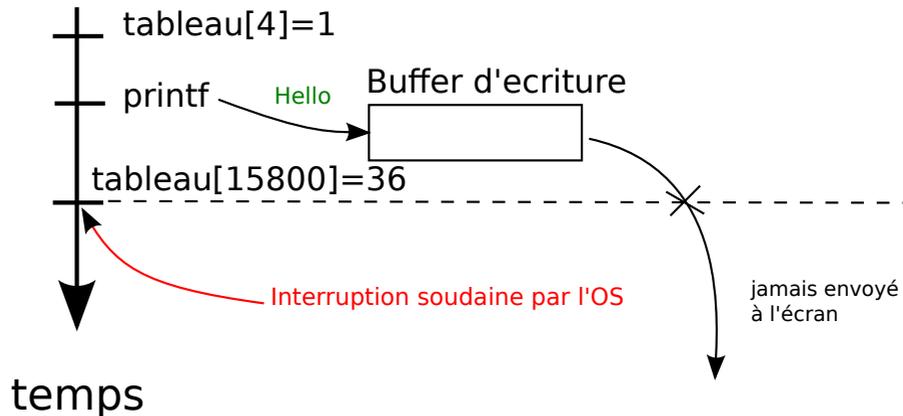
# Debug par affichage: printf

Attention à l'effet du tampon d'écriture

```
int main()
{
  int tableau[5];
  tableau[4]=1;
  printf("Hello");
  tableau[15800]=36;
  printf("World");
  return 0;
}
```

```
./mon_executable
> Segmentation fault
```

Pourtant seg-fault à la ligne suivante!!



# Debug par affichage: printf

Attention à l'effet du tampon d'écriture

```
int main()
{
    int tableau[5];
    tableau[4]=1;
    printf("Hello");
    tableau[15800]=36;
    printf("World");
    return 0;
}
```

```
./mon_executable
> Segmentation fault
```

Pourtant seg-fault à la ligne suivante!!



Solution: Toujours vider le buffer (flush)

- Retour à la ligne `"\n"`  
(écran uniquement)
- commande `fflush(stdout);`

```
int main()
{
    int tableau[5];
    tableau[4]=1;
    printf("Hello\n");
    tableau[15800]=36;
    printf("World\n");
    return 0;
}
```

déduit: seg-fault entre  
"Hello" et "World"

```
./mon_executable
> Hello
> Segmentation fault
```

# Debug par affichage: printf

## Avantages:

- + léger, rapide  
(pas d'outils externes)
- + rapide
- + execution et debug du vrai code  
(pas d'émulation)

## Inconvénients:

- Nécessite de bien connaître le code
- Nécessite de pré-localiser l'erreur
- Précaution lors de l'utilisation de données nombreuses

## Synthèse:

Méthode rapide lorsque l'on travaille sur son propre code (vérification d'une boucle, d'un pointeur, etc.)

Peu adapté pour le debug d'un code externe.

# Méthode de debug

Méthode manuelle (printf)

→ **Debugger à points d'arrêts**

Détecteur fuites mémoires

# Debug par debugger : semi-automatique

```
1 #include <stdio.h>
2
3 struct employe
4 {
5     char *nom;
6     int *paye;
7     char *service;
8 };
9
10 int main()
11 {
12
13     char *nom[]={ "Bertrand", "Bernard", "Renaud", "Simon" };
14     int paye[]={ 1200, 1500, 1800 };
15     char *service[]={ "qualitee", "marketing", "R&D", "direction" };
16
17     struct employe employe_0={ nom[2], &paye[0], service[1] };
18     struct employe employe_1={ nom[1], &paye[0], service[0] };
19     struct employe employe_2={ nom[0], &paye[2], service[2] };
20     struct employe employe_3; employe_3.nom=nom[3];
21
22     int cout_total=0;
23     cout_total=*employe_0.paye+
24         *employe_1.paye+
25         *employe_2.paye+
26         *employe_3.paye;
27
28
29     return 0;
30 }
31
```

cout_total	0	int
▼ employe_0		employe
nom		char *
paye	0x0	int *
service	"H\211l\$ØL\...	char *
▼ employe_1		employe
nom		char *
paye	1690667336...	int
service		char *
▼ employe_2		employe
nom	"Á"	char *
paye	-148758528...	int
service	"°A³÷ÿ\177"	char *
▼ employe_3		employe
nom	"\001"	char *
paye	1768709983...	int
service		char *
▼ nom	@0x7fffffff1f0	char *[4]
[0]	"Bertrand"	char *
[1]	"Bernard"	char *
[2]	"Renaud"	char *
[3]	"Simon"	char *
▼ paye	@0x7fffffff2...	int [3]
[0]	1200	int
[1]	1500	int
[2]	1800	int
▼ service	@0x7fffffff2...	char *[4]
[0]	"qualitee"	char *
[1]	"marketing"	char *
[2]	"R&D"	char *
[3]	"direction"	char *

point d'arrêt

variables + valeurs

# Debug par debugger : semi-automatique

```
cout_total 0 int
employee_0 employee
  nom char *
  paye int *
  service "H2111$0L... char *
employee_1 employee
  nom char *
  paye 1690667336... int
  service char *
employee_2 employee
  nom "A" char *
  paye -148758528... int
  service "A**y177" char *
employee_3 employee
  nom "001" char *
  paye 1768709983... int
  service char *
nom char *
  @0x7ffffff1f0 char *[4]
  [0] "Bertrand" char *
  [1] "Bernard" char *
  [2] "Renaud" char *
  [3] "Simon" char *
paye int
  [0] @0x7ffffff2... int [3]
  [1] 1200 int
  [2] 1500 int
  [3] 1800 int
service char *
  @0x7ffffffe2... char *[4]
  [0] "qualitee" char *
  [1] "marketing" char *
  [2] "R&D" char *
  [3] "direction" char *
```

```
1 #include <stdio.h>
2
3 struct employe
4 {
5     char *nom;
6     int *paye;
7     char *service;
8 };
9
10 int main()
11 {
12
13     char *nom[]={"Bertrand", "Bernard", "Renaud", "Simon"};
14     int paye[]={1200,1500,1800};
15     char *service[]{"qualitee", "marketing", "R&D", "direction"};
16
17     struct employe employe_0={nom[2], &paye[0], service[1]};
18     struct employe employe_1={nom[1], &paye[0], service[0]};
19     struct employe employe_2={nom[0], &paye[2], service[2]};
20     struct employe employe_3; employe_3.nom=nom[3];
21
22     int cout_total=0;
23     cout_total=*employe_0.paye+
24         *employe_1.paye+
25         *employe_2.paye+
26         *employe_3.paye;
27
28
29     return 0;
30
31 }
```

```
employee_0 employe
  nom "Renaud" char *
  paye 1200 @0x7ff... int
  service "marketing" char *
```

# Debug par debugger : semi-automatique

```
cout_total 0 int
employe_0   int
  nom       char *
  paye      int *
  service   "H2111$0L... char *
employe_1   employe
  nom       char *
  paye      1690667336... int
  service   employe
employe_2   char *
  nom       "A" char *
  paye      -148758528... int
  service   "A*y(177" char *
employe_3   employe
  nom       "001" char *
  paye      1768709983... int
  service   char *
  nom       @0x7ffffff1f0 char *[4]
  [0]      "Bertrand" char *
  [1]      "Bernard" char *
  [2]      "Renaud" char *
  [3]      "Simon" char *
  paye     @0x7ffffff2... int [3]
  [0]      1200 int
  [1]      1500 int
  [2]      1800 int
  service  @0x7ffffffe2... char *[4]
  [0]      "qualitee" char *
  [1]      "marketing" char *
  [2]      "R&D" char *
  [3]      "direction" char *
```

```
1 #include <stdio.h>
2
3 struct employe
4 {
5     char *nom;
6     int *paye;
7     char *service;
8 };
9
10 int main()
11 {
12
13     char *nom[]={"Bertrand", "Bernard", "Renaud", "Simon"};
14     int paye[]={1200,1500,1800};
15     char *service[]={"qualitee", "marketing", "R&D", "direction"};
16
17     struct employe employe_0={nom[2], &paye[0], service[1]};
18     struct employe employe_1={nom[1], &paye[0], service[0]};
19     struct employe employe_2={nom[0], &paye[2], service[2]};
20     struct employe employe_3; employe_3.nom=nom[3];
21
22     int cout_total=0;
23     cout_total=*employe_0.paye+
24         *employe_1.paye+
25         *employe_2.paye+
26         *employe_3.paye;
27
28
29     return 0;
30
31 }
```

```
employe_1   employe
  nom       "Bernard" char *
  paye      1200 @0x7fff... int
  service   "qualitee" char *
```

# Debug par debugger : semi-automatique

```
cout_total 0          int
employe_0    nom      employe
              paye     char *
              service  "H211f0L... char *
employe_1    nom      employe
              paye     1690667336... char *
              service  "A" employe
employe_2    nom      char *
              paye     -148758528... int
              service  "A*y(177" char *
employe_3    nom      employe
              paye     "001" char *
              service  1768709983... char *
nom          @0x7ffffff1f0 char [4]
[0]         "Bertrand" char *
[1]         "Bernard" char *
[2]         "Renaud" char *
[3]         "Simon" char *
paye        @0x7ffffffe2... int [3]
[0]         1200 int
[1]         1500 int
[2]         1800 int
service     @0x7ffffffe2... char [4]
[0]         "qualitee" char *
[1]         "marketing" char *
[2]         "R&D" char *
[3]         "direction" char *
```

```
1  #include <stdio.h>
2
3  struct employe
4  {
5      char *nom;
6      int *paye;
7      char *service;
8  };
9
10 int main()
11 {
12
13     char *nom[]={"Bertrand", "Bernard", "Renaud", "Simon"};
14     int paye[]={1200,1500,1800};
15     char *service[]{"qualitee", "marketing", "R&D", "direction"};
16
17     struct employe employe_0={nom[2], &paye[0], service[1]};
18     struct employe employe_1={nom[1], &paye[0], service[0]};
19     struct employe employe_2={nom[0], &paye[2], service[2]};
20     struct employe employe_3; employe_3.nom=nom[3];
21
22     int cout_total=0;
23     cout_total=*employe_0.paye+
24         *employe_1.paye+
25         *employe_2.paye+
26         *employe_3.paye;
27
28
29     return 0;
30
31 }
```

```
employe_2
nom      "Bertrand" char *
paye     1800 @0x7ff... int
service  "R&D" char *
```

# Debug par debugger : semi-automatique

```
cout_total 0      int
employe_0     nom   employe
              paye  char*
              service "H2111$0L... char*
              nom   employe
              paye  1690667336... char*
              service "A" employe
              nom   "A" char*
              paye  -148758528... int
              service "A*y177" char*
employe_3     nom   employe
              paye  "001" char*
              service 1768709983... int
              nom   @0x7ffffffe1f0 char*[4]
              [0] "Bertrand" char*
              [1] "Bernard" char*
              [2] "Renaud" char*
              [3] "Simon" char*
              paye  @0x7ffffffe2... int [3]
              [0] 1200 int
              [1] 1500 int
              [2] 1800 int
              service @0x7ffffffe2... char*[4]
              [0] "qualitee" char*
              [1] "marketing" char*
              [2] "R&D" char*
              [3] "direction" char*
```

```
1 #include <stdio.h>
2
3 struct employe
4 {
5     char *nom;
6     int *paye;
7     char *service;
8 };
9
10 int main()
11 {
12
13     char *nom[]={"Bertrand", "Bernard", "Renaud", "Simon"};
14     int paye[]={1200,1500,1800};
15     char *service[]{"qualitee", "marketing", "R&D", "direction"};
16
17     struct employe employe_0={nom[2], &paye[0], service[1]};
18     struct employe employe_1={nom[1], &paye[0], service[0]};
19     struct employe employe_2={nom[0], &paye[2], service[2]};
20     struct employe employe_3; employe_3.nom=nom[3];
21
22     int cout_total=0;
23     cout_total=*employe_0.paye+
24         *employe_1.paye+
25         *employe_2.paye+
26         *employe_3.paye;
27
28
29     return 0;
30
31 }
```

```
employe_3
nom "Simon" employe
paye 1768709983... int
service char*
```

# Debug par debugger : semi-automatique

Outils de debugs par points d'arrets:



**gdb:** mode texte

[cs.brynmawr.edu/cs312/gdb-tutorial-handout.pdf](http://cs.brynmawr.edu/cs312/gdb-tutorial-handout.pdf)

**kdbg:**

```
1 4
5 char *non;
6 int *paye;
7 char *service;
8 };
9
10 int main()
11 {
12
13     char non[]={"Bertrand", "Bernard", "Renaud", "Simon"};
14     int paye[]={1200, 1500, 1800};
15     char service[]={"qualitee", "marketing", "R&D", "direction"};
16
17     struct employe employe_0={non[2], &paye[0], service[0]};
18     struct employe employe_1={non[1], &paye[1], service[1]};
19     struct employe employe_2={non[0], &paye[2], service[2]};
20     struct employe employe_3={non[3], &paye[3], service[3]};
21
22     int cout_total=0;
23     cout_total+=employe_0.paye+
24     *employe_1.paye+
25     *employe_2.paye+
26     *employe_3.paye;
27
28
29     return 0;
30
31
32
33
34
```

**QtCreator:**

```
1 4
2 int main()
3 {
4     char non[]={"Bertrand", "Bernard", "Renaud", "Si
5     int paye[]={1200, 1500, 1800};
6     char service[]={"qualitee", "marketing", "R&D
7
8     struct employe employe_0={non[2], &paye[0], serv
9     struct employe employe_1={non[1], &paye[1], serv
10    struct employe employe_2={non[0], &paye[2], serv
11    struct employe employe_3={non[3], &paye[3], serv
12
13    int cout_total=0;
14    cout_total+=employe_0.paye+
15    *employe_1.paye+
16    *employe_2.paye+
17    *employe_3.paye;
18
19
20    return 0;
21
22
23
24
25
```

# Debug par debugger : semi-automatique

## Avantage:

- + Visuelle, vision globale
- + debug données complexes  
(listes chaînées, graphes, ...)

## Inconvénient:

- Localisation préalable  
(fichier, lignes, variables)
- Debugger peut être lent

## Synthèse:

Idéale pour debugger les structures de données complexes (pointeurs, ...).

Moins adapté pour le debug d'un code externe.

# Méthode de debug

Méthode manuelle (printf)

Debugger à points d'arrêts

→ **Détecteur fuites mémoires**

# Debug par debugger : automatique

valgrind ./mon\_executable



```
int main()
{
    int *p;
    int u=0;

    *p=5;
    return 0;
}
```

```
==3048== Memcheck, a memory error detector
==3048== Copyright (C) 2002-2012, and GNU GPL'd, by Julian Seward et al.
==3048== Using Valgrind-3.8.0 and LibVEX; rerun with -h for copyright info
==3048== Command: ./a.out
==3048==
==3048== Use of uninitialised value of size 8
==3048==    at 0x4004BB: main (debug_04.c:10)
==3048==
==3048== HEAP SUMMARY:
==3048==    in use at exit: 0 bytes in 0 blocks
==3048==   total heap usage: 0 allocs, 0 frees, 0 bytes allocated
==3048==
==3048== All heap blocks were freed -- no leaks are possible
==3048==
==3048== For counts of detected and suppressed errors, rerun with: -v
==3048== Use --track-origins=yes to see where uninitialised values come from
==3048== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 2 from 2)
```

=> Autrement: non détectée en tant que seg-fault.

# Debug par debugger : automatique

## Gestion mémoire dynamique:

```
int main()
{
    int *p=NULL;
    p=malloc(10);

    return 0;
}
```

```
==3666== Memcheck, a memory error detector
==3666== Copyright (C) 2002-2012, and GNU GPL'd, by Julian Seward et al.
==3666== Using Valgrind-3.8.0 and LibVEX; rerun with -h for copyright info
==3666== Command: ./a.out
==3666==
==3666== HEAP SUMMARY:
==3666==   in use at exit: 10 bytes in 1 blocks
==3666== total heap usage: 1 allocs, 0 frees, 10 bytes allocated
==3666==
==3666== LEAK SUMMARY:
==3666==   definitely lost: 10 bytes in 1 blocks
==3666==   indirectly lost: 0 bytes in 0 blocks
==3666==   possibly lost: 0 bytes in 0 blocks
==3666==   still reachable: 0 bytes in 0 blocks
==3666==   suppressed: 0 bytes in 0 blocks
==3666== Rerun with --leak-check=full to see details of leaked memory
==3666==
==3666== For counts of detected and suppressed errors, rerun with: -v
==3666== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 2 from 2)
```

très difficile à détecter autrement!

```
int main()
{
    int *p=NULL;
    p=malloc(10);

    free(p);

    return 0;
}
```

```
==3555== Memcheck, a memory error detector
==3555== Copyright (C) 2002-2012, and GNU GPL'd, by Julian Seward et al.
==3555== Using Valgrind-3.8.0 and LibVEX; rerun with -h for copyright info
==3555== Command: ./a.out
==3555==
==3555== HEAP SUMMARY:
==3555==   in use at exit: 0 bytes in 0 blocks
==3555== total heap usage: 1 allocs, 1 frees, 10 bytes allocated
==3555==
==3555== All heap blocks were freed -- no leaks are possible
==3555==
==3555== For counts of detected and suppressed errors, rerun with: -v
==3555== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 2 from 2)
```

```
int main()
{
    int *p=NULL;
    p=malloc(10);

    free(p);
    free(p);

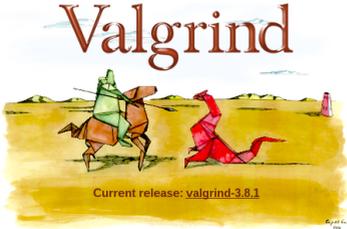
    return 0;
}
```

```
==3615== Memcheck, a memory error detector
==3615== Copyright (C) 2002-2012, and GNU GPL'd, by Julian Seward et al.
==3615== Using Valgrind-3.8.0 and LibVEX; rerun with -h for copyright info
==3615== Command: ./a.out
==3615==
==3615== Invalid free() / delete / delete[] / realloc()
==3615==   at 0x4C2ACBC: free (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==3615==   by 0x480571: main (debug 05.c:11)
==3615==   Address 0x51da040 is 0 bytes inside a block of size 10 free'd
==3615==   at 0x4C2ACBC: free (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==3615==   by 0x480565: main (debug 05.c:10)
==3615==
==3615== HEAP SUMMARY:
==3615==   in use at exit: 0 bytes in 0 blocks
==3615== total heap usage: 1 allocs, 2 frees, 10 bytes allocated
==3615==
==3615== All heap blocks were freed -- no leaks are possible
==3615==
==3615== For counts of detected and suppressed errors, rerun with: -v
==3615== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 2 from 2)
```

# Debug par debugger : automatique

Outils:

Valgrind



émule le processeur+registres  
=> vérifie l'utilisation mémoire

Fonctionnement:

```
valgrind ./mon_executable
```

↑ compilé en mode debug



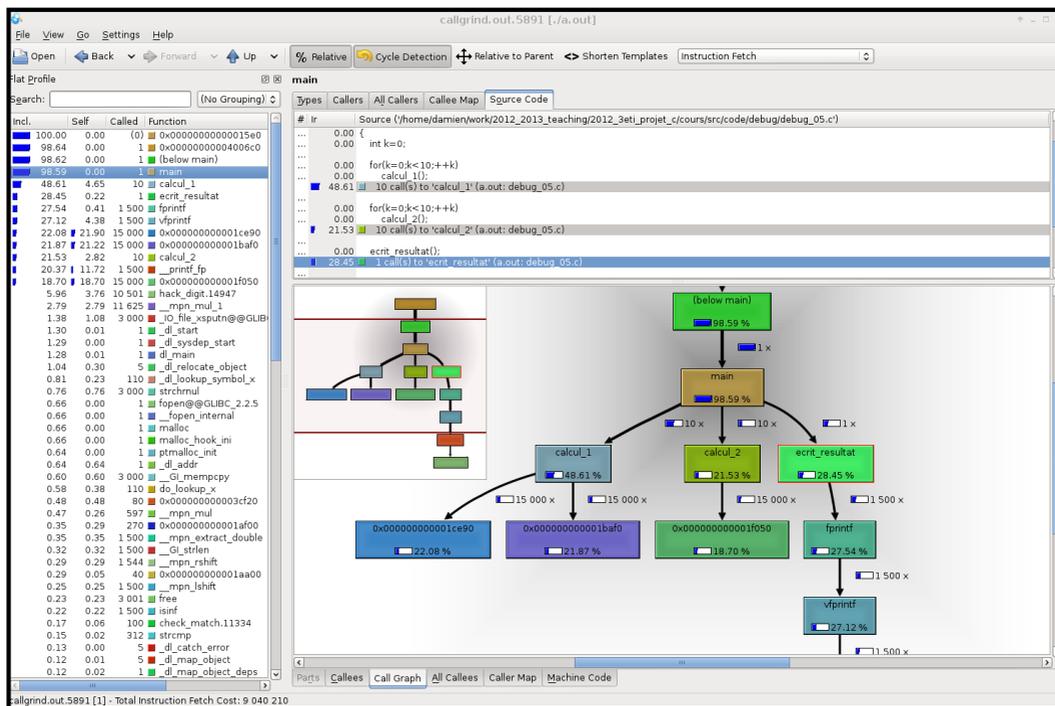
Très utile pour du controle final (*sanity check*)

# Debug par debugger : automatique

Possibilité de profiling



```
valgrind --tool=cachegrind ./mon_executable  
kcachegrind callgrind.out.xxxx
```



# Debug par debugger : automatique

## Avantages:

- + détection automatique erreurs
- + permet d'assurer l'absence de fuites mémoire

## Inconvénients:

- uniquement debug erreurs mémoires
- code émulé  
(plus lent, comportement différent possible)
- détection d'erreurs librairies externes

## Synthèse:

Très adapté pour la détection de fuite mémoire pour un grand code. Il n'est pas nécessaire d'être familier du code.

Toujours utiliser en tant que "*sanity check*" avant de délivrer le code.

Sortie parfois difficile à interpréter, surtout si d'autres librairies possèdent des fuites mémoires elle-même.

# Tests

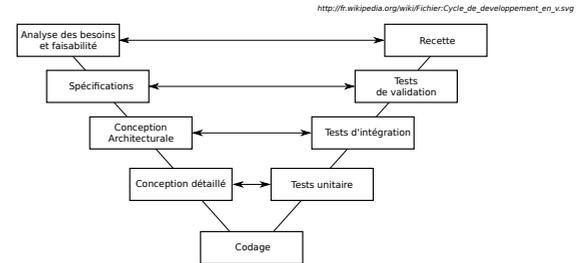
Principe

Méthodologies de tests unitaires

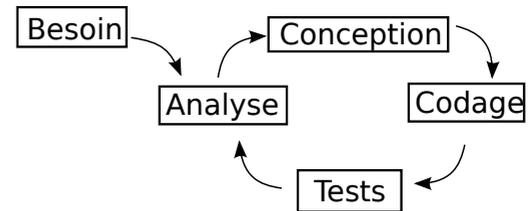
# Methodes de génie logiciel

## Grandes méthodologies

### Approche en V

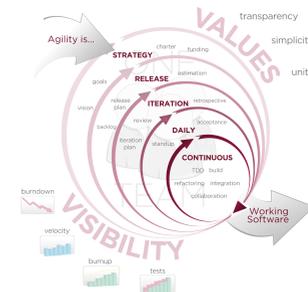


### Approche itérative



### Approche Agile

Extreme Programming (XP)  
Agle Unified Process (AUP)  
Dynamic Systems Development Methode (DSDM)  
Feature Driven Development (FDD)  
Scrum  
Kanban  
...



# Role des tests

Importance capitale des tests !

Code de qualité =>

Tests permettant de certifier le code

Tester tout au long du developpement

But: détecter les défauts le plus tôt possible

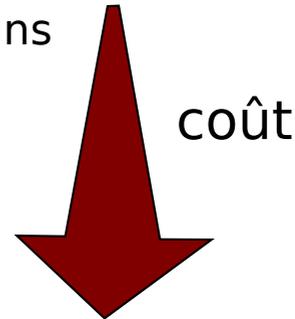
Analyse des besoins

Design

Code

Tests intégrations

Chez le client



# Catégories des tests

## Tests unitaires

Test de fonctionnalités de manière unitaire.

(plusieurs tests par fonctions: très nombreux)

---

## Tests d'intégrations

Tests la bonne execution d'un ensemble de fonctionnalités

(plusieurs tests par grande fonctionnalité)

---

## Tests système

Test finale du système au complet

(quelques tests par logiciel)

# Principe des tests

Dans un logiciel de qualité

=> Toute les fonctions doivent être testée de manière unitaire

Testez un maximum vos fonctions!

Les cas de bases

Les cas avancés

Les cas qui fonctionnent

Les cas qui ne fonctionnent pas

Les cas critiques/particuliers

Le plus  
exhaustivement  
possible

# Tests

Principe

→ **Méthodologies de tests unitaires**

# Tests unitaires

## Tests par échantillonnage

```
int somme(int a,int b);
```

```
int test_somme()
```

```
{
```

```
    if (somme(1,1) != 2)
```

```
        ERREUR_TEST;
```

```
    if (somme(5,8) != 13)
```

```
        ERREUR_TEST;
```

```
    if (somme(-5,8) != 3)
```

```
        ERREUR_TEST;
```

```
    if (somme(-10,-5) != -15)
```

```
        ERREUR_TEST;
```

```
}
```

cas base

cas normal

nombre négatif

nombre négatif x2

# Tests unitaires

## Tests par échantillonnage (avec cas particuliers)

```
#define GRAND_ENTIER 2147483647
int somme(int a, int b, int *code_erreur);

int test_somme()
{
    int code_erreur=0;
    if (somme(1,1,&code_erreur)!=2 || code_erreur!=0)
        ERREUR_TEST;
    if (somme(5,8,&code_erreur)!=13 || code_erreur!=0)
        ERREUR_TEST;
    if (somme(-5,8,&code_erreur)!=3 || code_erreur!=0)
        ERREUR_TEST;
    if (somme(-10,-5,&code_erreur)!=-15 || code_erreur!=0)
        ERREUR_TEST;
    if (somme(0,1,&code_erreur)!=1 || code_erreur!=0)
        ERREUR_TEST;
    if (somme(0,0,&code_erreur)!=0 || code_erreur!=0)
        ERREUR_TEST;

    int code_erreur=0;
    somme(GRAND_ENTIER,1,&code_erreur);
    if (code_erreur!=ERREUR_DEPASSEMENT)
        ERREUR_TEST;

    int code_erreur=0;
    somme(1,GRAND_ENTIER,&code_erreur);
    if (code_erreur!=ERREUR_DEPASSEMENT)
        ERREUR_TEST;

    int code_erreur=0;
    somme(-GRAND_ENTIER,2,&code_erreur);
    if (code_erreur!=ERREUR_DEPASSEMENT)
        ERREUR_TEST;

    int code_erreur=0;
    somme(2,-GRAND_ENTIER,&code_erreur);
    if (code_erreur!=ERREUR_DEPASSEMENT)
        ERREUR_TEST;

    return OK_TEST;
}
```

test avec le plus grand entier

=> Nécessite de gérer une erreur

=> implémentation + complexe que return a+b;

# Tests unitaires

## Tests automatiques

```
int somme(int a, int b);

int test_somme()
{
    int nombre_max=9999999;

    int k1=-nombre_max;
    int k2=-nombre_max;
    for(k1=0;k1<nombre_max;++k1)
        for(k2=0;k2<nombre_max;++k2)
            if(somme(k1,k2) != k1+k2)
                ERREUR_TEST;

    return TEST_OK;
}
```

=> Permet d'être plus exhaustif

# Tests unitaires

## Gestion de l'affichage

ex. de mauvais affichage:

```
printf("%d\n", somme(5,8));  
printf("%d\n", somme(0,1));  
printf("%d\n", somme(-1,0));  
printf("%d\n", somme(9999,5));  
printf("%d\n", somme(845,1));  
printf("%d\n", somme(2147483647,1));
```



```
13  
1  
-1  
10004  
846  
-2147483648
```

long et difficile à interpréter!

But=> faciliter l'analyse pour pouvoir  
lancer les tests plusieurs fois.  
Préférez solution type: **OK** / **KO**

# Tests unitaires

## Gestion de l'affichage

### Amélioration:

```
#define TEST_ECHOUÉ printf("Test unitaire échoué: l.%d fichier %s\n", __LINE__, __FILE__)

int somme(int a, int b);

void test_somme()
{
    if(somme(1,1)!=2)
        TEST_ECHOUÉ;
    if(somme(1,5)!=6)
        TEST_ECHOUÉ;
    if(somme(0, -1)!=-1)
        TEST_ECHOUÉ;
    if(somme(-4, -12)!=-16)
        TEST_ECHOUÉ;
}

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

int somme(int a, int b)
{
    if(b<0) a+=1;
    return a+b;
}
```

Affichage uniquement en cas d'erreur.  
Indique: ligne+fichier

Test unitaire échoué: l.13 fichier test\_unit.c  
Test unitaire échoué: l.15 fichier test\_unit.c

# Tests unitaires

## Gestion de l'affichage

### Autre possibilité: assert

```
int somme(int a, int b);  
  
void test_somme()  
{  
    assert(somme(1,1)==2);  
    assert(somme(1,5)==6);  
    assert(somme(0, -1)==-1);  
    assert(somme(-4, -12)==-16);  
}  
  
int main()  
{  
    test_somme();  
    return 0;  
}  
  
int somme(int a, int b)  
{  
    if(b<0) a+=1;  
    return a+b;  
}
```

Affichage uniquement en cas d'erreur.

S'arrête à la première erreur

Indique: ligne+fichier+condition en erreur

```
a.out: test_03.c:11: test_somme: Assertion `somme(0,-1)==-1' failed.  
Aborted
```

## Gestion de l'affichage

### Affichage complet

```
#define UNIT_TEST(commande) \
    if( !(commande) ) \
        {printf("Test echoue: [%s], l.%d, fonction %s, fichier %s\n", \
                #commande, __LINE__, __FUNCTION__, __FILE__); } \
    else \
        {printf("Test OK: [%s]\n", #commande);} \

int somme(int a, int b);

void test_somme()
{
    UNIT_TEST(somme(1,1)==2);
    UNIT_TEST(somme(1,5)==6);
    UNIT_TEST(somme(10,5)==15);
    UNIT_TEST(somme(-1,5)==4);
    UNIT_TEST(somme(-1,-5)==-6);
    UNIT_TEST(somme(-100,5)==-95);
}

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

int somme(int a, int b)
{
    if(b<0) a+=1;
    return a+b;
}
```

```
Test OK: [somme(1,1)==2]
Test OK: [somme(1,5)==6]
Test OK: [somme(10,5)==15]
Test OK: [somme(-1,5)==4]
Test echoue: [somme(-1,-5)==-6], l.21, fonction test_somme, fichier test_03.c
Test OK: [somme(-100,5)==-95]
```

# Tests unitaires



## Synthèse:

L'écriture des tests est un travail long  
Souvent + long que le code lui même

Ecriture  
+  
**Reflexion** sur choix  
pertinent de tests

Mais: Gain de temps au final

Moins de debug  
Pas de retour en arrière

=> Code de qualité

Ecrit 1x  
Utilisé un grand nombre de fois

Un bon test = test qui se réutilise  
test dont le résultat est rapidement analysé  
test qui couvre un grand nombre de cas  
test qui vérifie les cas critiques (corrects et incorrects)

# Tests unitaires

Outils de tests unitaires:

Macros C (bon cas d'utilisation de macros)

CUnit

<http://cunit.sourceforge.net/>

```
/* Simple test of fprintf().
 * Writes test data to the temporary file and checks
 * whether the expected number of bytes were written.
 */
void testFPRINTF(void)
{
    int i1 = 10;

    if (NULL != temp_file) {
        CU_ASSERT(0 == fprintf(temp_file, ""));
        CU_ASSERT(2 == fprintf(temp_file, "Q\n"));
        CU_ASSERT(7 == fprintf(temp_file, "i1 = %d", i1));
    }
}
```

CUnit - A Unit testing framework for C.  
<http://cunit.sourceforge.net>

File Name	Condition	Line Number	Status
CUnitTest.c	CU_ASSERT_EQUAL(0,3)	37	Failed
CUnitTest.c	CU_ASSERT_STRING_EQUAL("string #1","string #2")	47	Failed
CUnitTest.c	CU_ASSERT_EQUAL(1,3)	37	Failed

Cumulative Summary for Run				
Type	Total	Run	Succeeded	Failed
Suites	4	3	-NA-	2
Test Cases	13	10	7	3
Assertions	10	10	7	3

tableau de bord des résultats

# Design d'API

# Design code de logiciel

Fonction: 2 utilités

1. Effectuer une tache complexe pour le programmeur
2. Proposer une interface simple pour l'*utilisateur*

algorithmique

ex.

```
float calcul_volume_seve(const struct *tronc,  
                        const struct *feuillage,  
                        int age);
```

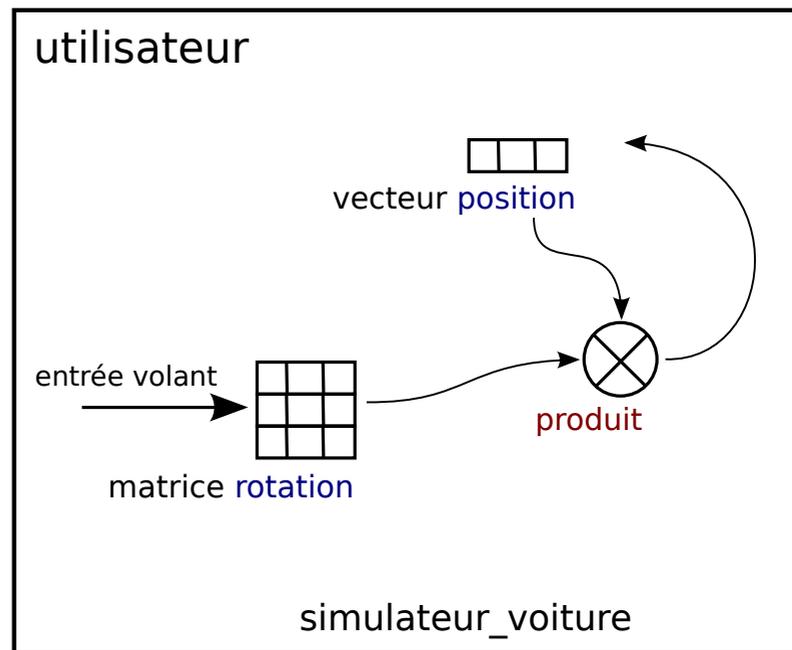
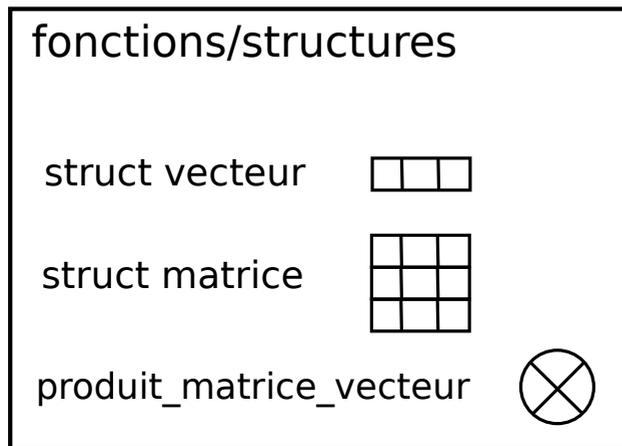
design API

```
struct foret genere_foret();
```

# Design code de logiciel

2. Proposer une interface simple pour l'utilisateur

Personne (programmeur) qui va utiliser les éléments du code en tant que boîte noire pour réaliser une tâche plus complexe.



# Design code de logiciel

## **Design d'API =**

comment réaliser des fonctions/structures utilisable aisément par l'utilisateur

API = Application Programming Interface

=> L'interface permettant d'utiliser le code de l'application

En C: Design d'API = Design des en-têtes de fonctions + structs !

→ L'API est donnée par les fichiers .h

**Note:** Toutes les en-têtes (fichiers .h) ne font pas partie de l'API

# Design API

Règles standards d'utilisation:

**l'API doit être fixe au cours du temps**  
(l'implémentation peut changer, pas l'API)

changement local OK



changement de  
tous les codes utilisant  
cette librairie KO



# Design API

Règles standards d'utilisation: **l'API doit être fixe au cours du temps**  
(l'implémentation peut changer, pas l'API)

exemple:

```
struct vecteur;  
  
void vecteur_init(struct vecteur* v, float x, float y, float z);  
float vecteur_norme(const struct vecteur* v);
```

*interface*

```
int main()  
{  
    struct vecteur v0;  
    struct vecteur v1;  
    struct vecteur v2;  
  
    vecteur_init(&v0, 0, 1, 0);  
    vecteur_init(&v1, 1, 1, 0);  
    vecteur_init(&v2, 0, 1, 2);  
  
    float n0=vecteur_norme(&v0);  
    float n1=vecteur_norme(&v1);  
    float n2=vecteur_norme(&v2);  
  
    return 0;  
}
```

*utilisation*

# Design API

Règles standards d'utilisation: **l'API doit être fixe au cours du temps**  
(l'implémentation peut changer, pas l'API)

exemple:

```
struct vecteur;  
  
void vecteur_init(struct vecteur* v, float x, float y, float z);  
float vecteur_norme(const struct vecteur* v);
```

*interface*

**+ L'interface et l'utilisation sont identiques!!!**

```
int main()  
{  
    struct vecteur v0;  
    struct vecteur v1;  
    struct vecteur v2;  
  
    vecteur_init(&v0, 0, 1, 0);  
    vecteur_init(&v1, 1, 1, 0);  
    vecteur_init(&v2, 0, 1, 2);  
  
    float n0=vecteur_norme(&v0);  
    float n1=vecteur_norme(&v1);  
    float n2=vecteur_norme(&v2);  
  
    return 0;  
}
```

*utilisation*

```
struct vecteur  
{  
    float x;  
    float y;  
    float z;  
};  
  
void vecteur_init(struct vecteur* v, float x, float y, float z)  
{  
    v->x=x;  
    v->y=y;  
    v->z=z;  
}  
  
float vecteur_norme(const struct vecteur* v)  
{  
    return sqrt(v->x*v->x+v->y*v->y+v->z*v->z);  
}
```

*implémentation 1*

```
struct vecteur  
{  
    float donnees[3];  
};  
  
void vecteur_init(struct vecteur* v, float x, float y, float z)  
{  
    v->donnees[0]=x;  
    v->donnees[1]=y;  
    v->donnees[2]=z;  
}  
  
float vecteur_norme(const struct vecteur* v)  
{  
    float n2=v->donnees[0]*v->donnees[0]+  
            v->donnees[1]*v->donnees[1]+  
            v->donnees[2]*v->donnees[2];  
    return sqrt(n2);  
}
```

*implémentation 2*

# Design API

Règles standards d'utilisation: **l'API doit être fixe au cours du temps**  
(l'implémentation peut changer, pas l'API)

exemple:

```
struct vecteur;  
void vecteur_init(struct vecteur* v, float x, float y, float z);  
float vecteur_norme(const struct vecteur* v);
```

*interface 1*

```
int main()  
{  
    struct vecteur v0;  
    struct vecteur v1;  
    struct vecteur v2;  
  
    vecteur_init(&v0, 0, 1, 0);  
    vecteur_init(&v1, 1, 1, 0);  
    vecteur_init(&v2, 0, 1, 2);  
  
    float n0=vecteur_norme(&v0);  
    float n1=vecteur_norme(&v1);  
    float n2=vecteur_norme(&v2);  
  
    return 0;  
}
```

*utilisation*

```
struct vecteur;  
void vecteur_init(struct vecteur* v, float donnees[3]);  
void vecteur_norme(const struct vecteur* v, float *norme);
```



*interface 2*

- pas compatible!!!  
Modification ensemble du programme  
nécessaire!

# Design API

Règles standards d'utilisation:

**l'API doit être de *haut niveau***  
(par rapport à son rôle)

lisibilité  
(viser l'auto-documentation)

simplicité d'utilisation

# Design API

Règles standards d'utilisation:

L'API doit être de *haut niveau*  
(par rapport à son rôle)

*API*

```
#define TAILLE_DATE 12
#define TAILLE_NOM 128
#define TAILLE_DONNEES 4096

struct ip
{
    int donnees[4];
};
struct message
{
    struct ip ip_source;
    struct ip ip_destination;
    char auteur[TAILLE_NOM];
    char donnees[TAILLE_DONNEES];
    char date[TAILLE_DATE];
};
void message_initialise(struct message* mon_message,
                       struct ip ip_source,
                       struct ip ip_destination,
                       char auteur[],
                       char donnees[],
                       char date[]);

struct modem;
void modem_connecte_internet(struct modem* mon_modem);
void modem_fin_connection(struct modem* mon_modem);

struct message recoit_message(const struct modem* mon_modem);
void envoit_message(const struct modem* mon_modem,
                    const struct message* mon_message);
```



*Utilisation*

```
int main()
{
    struct message mon_message;

    struct ip ip_source={192,168,34,28};
    struct ip ip_destination={192,165,28,12};
    char mon_nom[]="Claude Francois";
    char mon_message[]="Alexandrie, Alexandra";
    char aujourdhui[]="18/03/1973";

    struct message mon_message;
    message_initialise(&mon_message,
                      ip_source,
                      ip_destination,
                      mon_nom,
                      mon_message,
                      aujourdhui);

    struct modem mon_modem;

    modem_connecte_internet(&mon_modem);
    envoit_message(&mon_modem,&mon_message);
    modem_fin_connection(&mon_modem);

    return 0;
}
```



Haut niveau = précis,lisible



# Design API



Règles standards d'utilisation:

**L'API doit être de *haut niveau***  
(par rapport à son rôle)

La notion de haut/bas niveau dépend du contexte/utilisation.

Haut-niveau  $\sim$  niveau d'abstraction modélisant l'objet/action réelle sans autres détails d'implémentations techniques.

```
struct ip_source=struct ip={192,168,34,28};
struct ip_dest_ecran_central=struct ip={192,157,05,01};
struct ip_dest_ecran_droit=struct ip={192,157,05,02};
struct ip_dest_ecran_gauche=struct ip={192,157,05,03};

struct message msg_1={ip_source,
                      ip_dest_ecran_gauche,
                      "inutile",
                      "temps restant: [5:00]\n",
                      "21/04/2012"};
struct message msg_2={ip_source,
                      ip_dest_ecran_central,
                      "inutile",
                      "OM VS PSG\n 0 - 2\n",
                      "21/04/2012"};
struct message msg_3={ip_source,
                      ip_dest_ecran_droit,
                      "inutile",
                      "Applaudissez\n",
                      "21/04/2012"};

struct modem mon_modem;
modem_connecte_internet(&mon_modem);
envoie_message(&mon_modem, &msg_1);
envoie_message(&mon_modem, &msg_2);
envoie_message(&mon_modem, &msg_3);
modem_fin_connection(&mon_modem);
```

*trop bas niveau:*  
*inutile d'exposer l'utilisation d'internet*

```
struct ecran;
void ecran_initialise(struct ecran* mon_ecran, char identificateur[]);
void ecrit(const struct* ecran, char message[]);
```

*nouvelle API au dessus de la gestion de paquets et d'IP*

```
struct ecran_central;
struct ecran_lateral_droit;
struct ecran_lateral_gauche;

ecran_initialise(&ecran_central, "centre");
ecran_initialise(&ecran_lateral_gauche, "lateral gauche");
ecran_initialise(&ecran_lateral_droit, "lateral droit");

ecrit(&ecran_lateral_gauche, "temps restant: [5:00]\n");
ecrit(&ecran_central, "OM VS PSG\n 0 - 2\n");
ecrit(&ecran_lateral_droit, "Applaudissez\n");
```

*haut niveau:*  
*on ne gère que l'identifiant des écrans et le texte*

Exemple: affichage d'un message sur 3 écrans lors d'un match de foot

# Design API

Règles standards d'utilisation:

*l'API doit être de **haut niveau***  
(par rapport à son rôle)

**Rappel:** Un bon code = code **lisible**



= code qui **cache sa complexité**  
= code qui propose de manipuler des  
**abstractions** aisément

# Design API

Règles standards d'utilisation:

Chaque bloc de l'API doit **minimiser**:

- \* l'exposition de son implémentation
- \* le nombre de fonctions les manipulant
- \* les interactions avec les autres blocs



**Truc:** Pour vérifier l'organisation de votre interface, représenter la (avec les interactions) sous forme de schéma bloc.  
Le schéma est-il simple, logique, lisible?

# Design API

Règles standards d'utilisation:

minimiser l'exposition de son implémentation

Exemple: accès coordonnées d'un vecteur:

vecteur

```
float x  
float y  
float z
```

```
x=vecteur.x  
y=vecteur.y  
z=vecteur.z
```

vecteur

```
float d[3]
```

```
x=vecteur.d[0]  
y=vecteur.d[1]  
z=vecteur.d[2]
```

vecteur

```
char d[12]
```

```
x=*(float*)(vecteur.d);  
y=*(float*)(vecteur.d+4);  
z=*(float*)(vecteur.d+8);
```

Implémentation non cachée.

Accès dépend du choix de la structure interne.

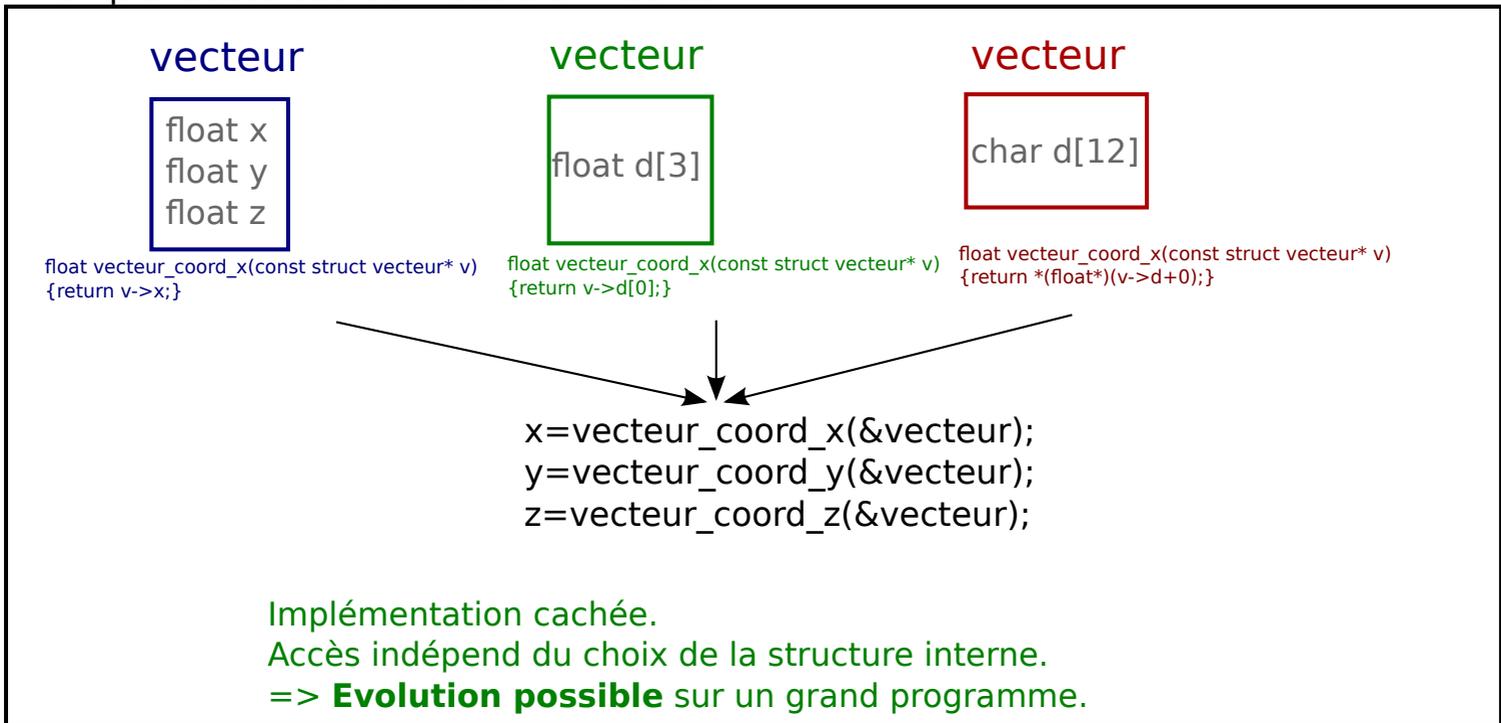
=> Evolution impossible sur un grand programme.

# Design API

Règles standards d'utilisation:

minimiser l'exposition de son implémentation

Exemple: accès coordonnées d'un vecteur:



# Design API

Règles standards d'utilisation:

minimiser l'exposition de son implémentation

**Note:** Pour les struct *conteneurs*

→ but est de contenir des données  
(vecteurs, tableaux, ...)

Opérations de *get/set* très classiques

Dénomination=**Accessors**

ex.

```
vecteur v;  
v.get_x();  
v.get_y(4.5);
```

```
tree b;  
b.get_trunc();
```

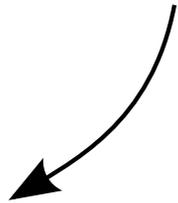
```
airplane a;  
a.get_reactor();  
a.set_weapon(weapon_1);
```

- => On ne sait pas comment sont stockées les données
- => On ne veut pas le savoir vue utilisateur (encapsulation)
- => Stockage interne peut être modifié sans changer l'API

# Design API

Règles standards d'utilisation:

minimiser le nombre de fonctions les manipulants



fonctionnalités précises  
niveau d'abstraction bien défini

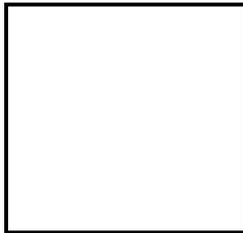
- + simple à gérer
- + simple à faire évoluer
- + augmente la cohérence

# Design API

Règles standards d'utilisation:

minimiser le nombre de fonctions les manipulants

struct voiture



voiture\_set/get\_carburant();  
voiture\_set/get\_puissance();  
voiture\_distance\_parcourus();  
voiture\_avance(int km);  
voiture\_avance\_paris();  
voiture\_avance\_marseille();  
voiture\_avance\_lyon();  
voiture\_entretien();  
voiture\_repare();

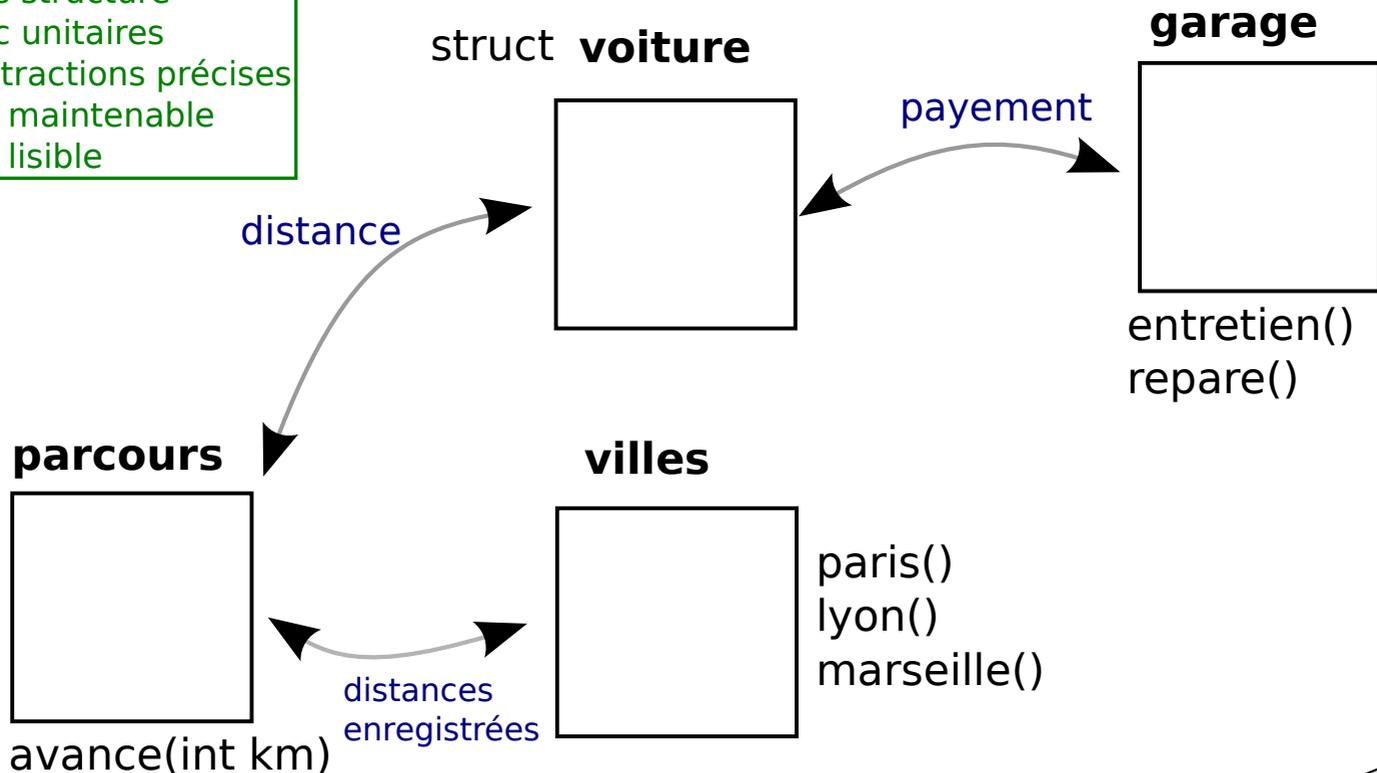
- trop de fonctions
- difficile à manipuler/faire évoluer

# Design API

Règles standards d'utilisation:

minimiser le nombre de fonctions les manipulant

plus structuré  
bloc unitaires  
abstractions précises  
=> maintenable  
=> lisible

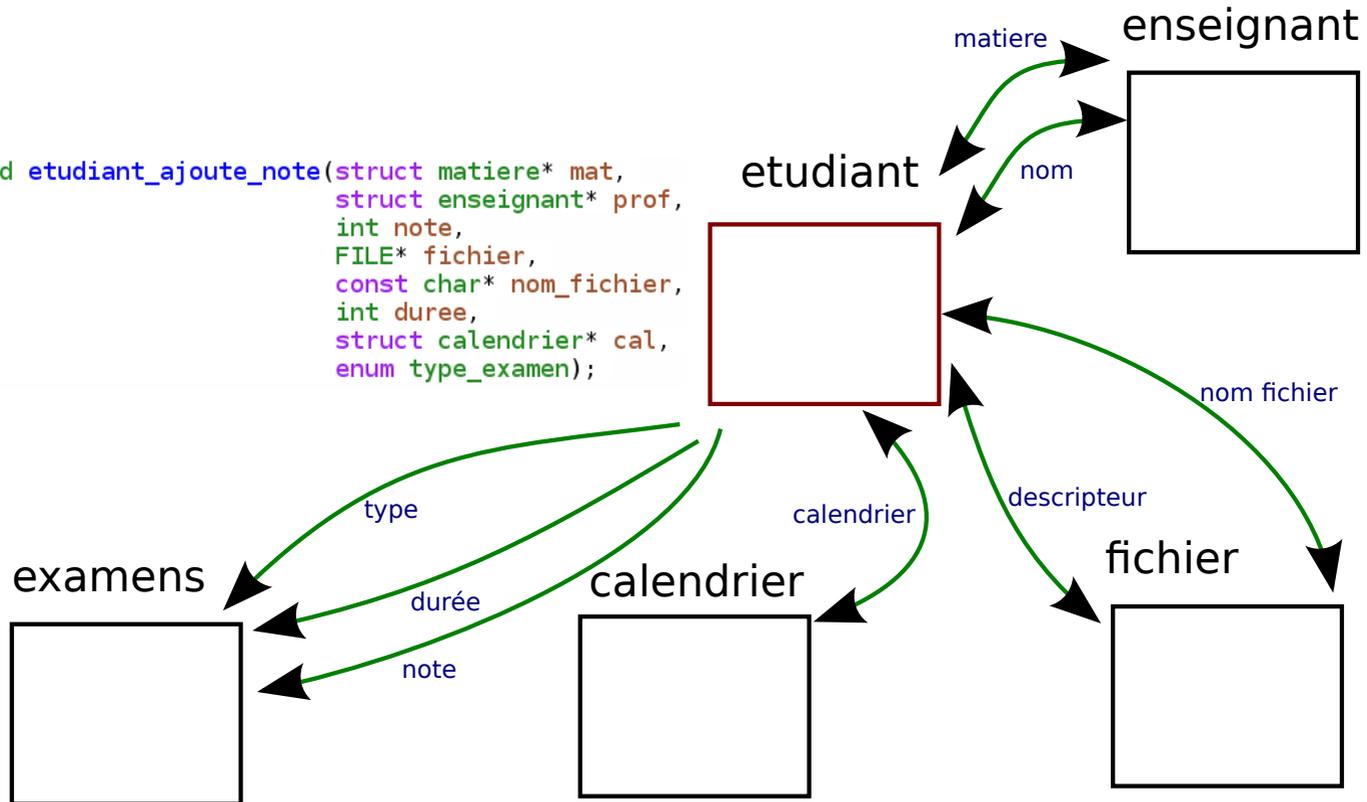


# Design API

Règles standards d'utilisation:

minimiser les interactions avec les autres blocs

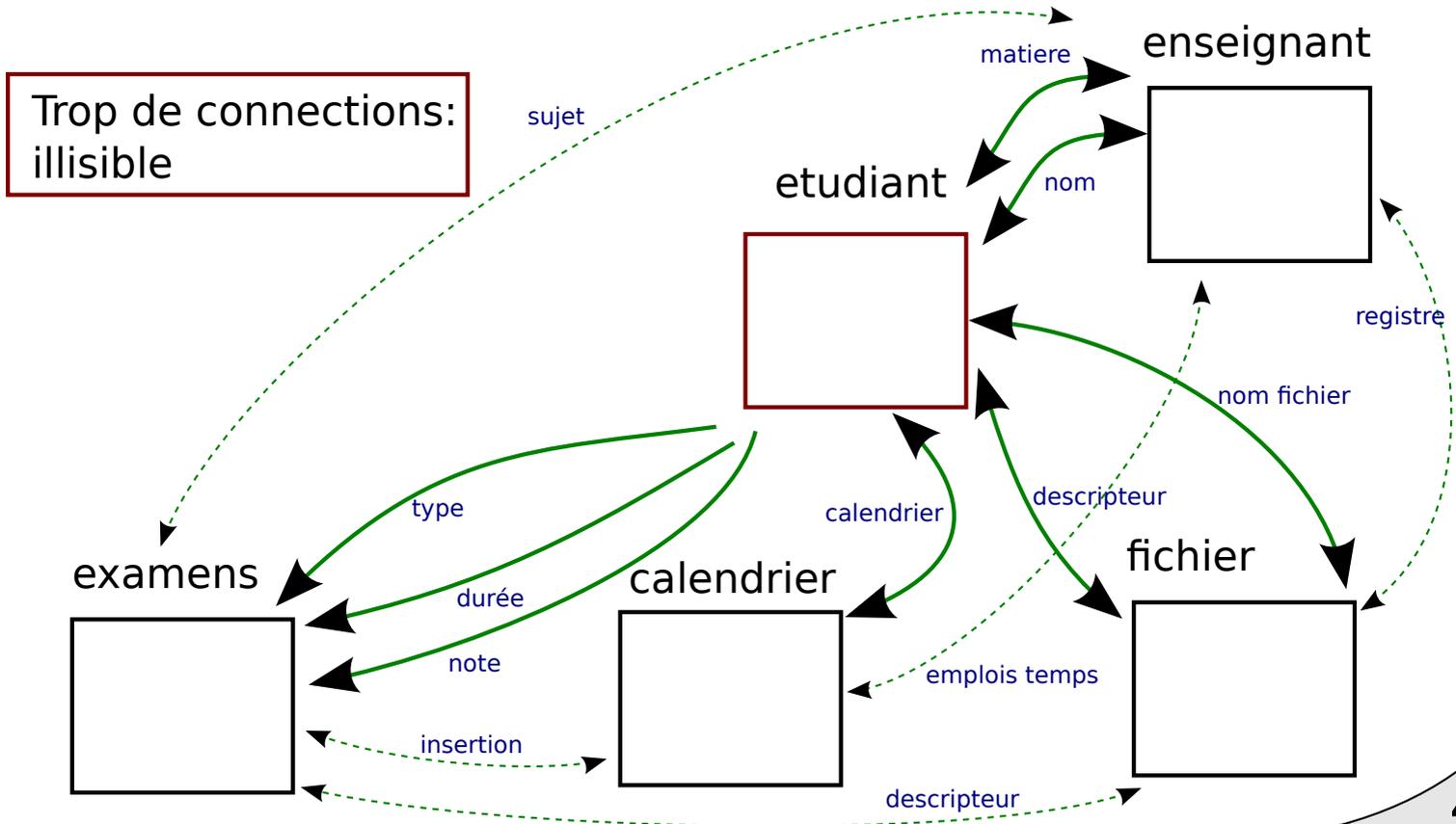
```
void etudiant_ajoute_note(struct matiere* mat,  
                          struct enseignant* prof,  
                          int note,  
                          FILE* fichier,  
                          const char* nom_fichier,  
                          int duree,  
                          struct calendrier* cal,  
                          enum type_examen);
```



# Design API

Règles standards d'utilisation:

minimiser les interactions avec les autres blocs

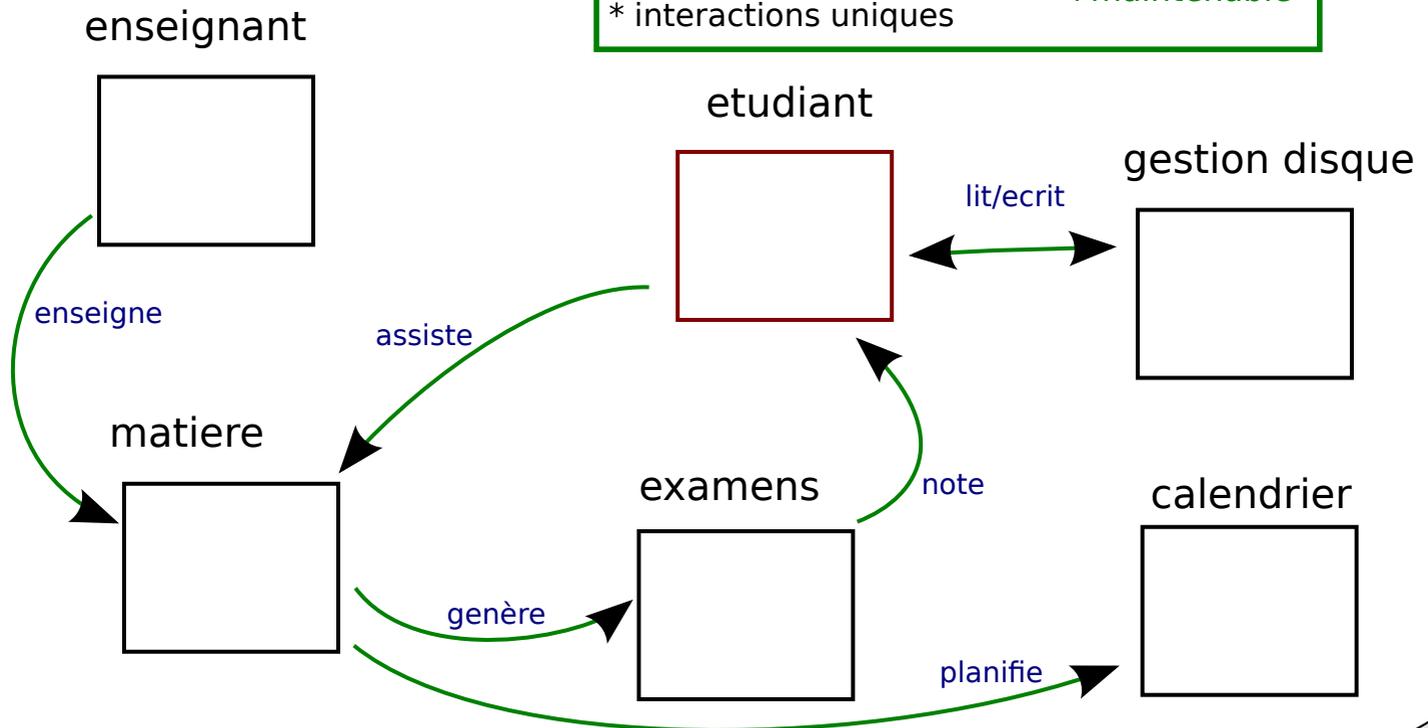


# Design API

Règles standards d'utilisation:

minimiser les interactions avec les autres blocs

\* flux directionnel  
\* moins d'interactions => +lisible  
\* interactions uniques +maintenable



Règles standards d'utilisation:

minimiser les interactions avec les autres blocs

Règles suggérées:

1 bloc communique avec au plus 3 autres blocs

Evitez les communication bi-directionnelles

Evitez les abstractions *omniscientes*

→ qui a accès à tout

# Developpement logiciel en C

## Software development in C

**Contact:** [damien.rohmer@cpe.fr](mailto:damien.rohmer@cpe.fr)

