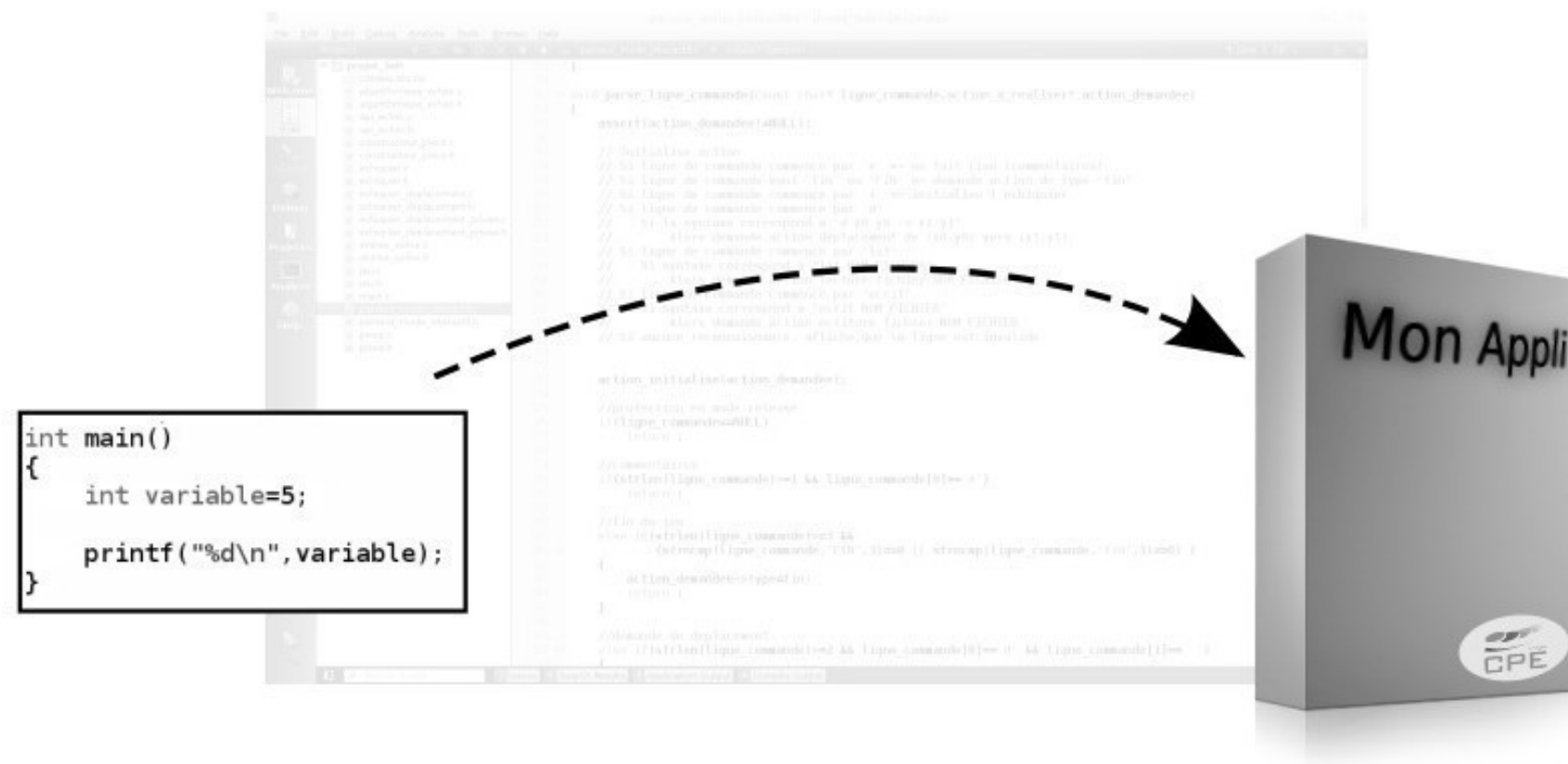


Developpement logiciel en C

Software development in C



000

Developpement logiciel en C

damien.rohmer@cpe.fr

6 Cours (2h)

6 séances projets (4h)

(36h)

6 travaux autonomie (3h)

+

18h révision

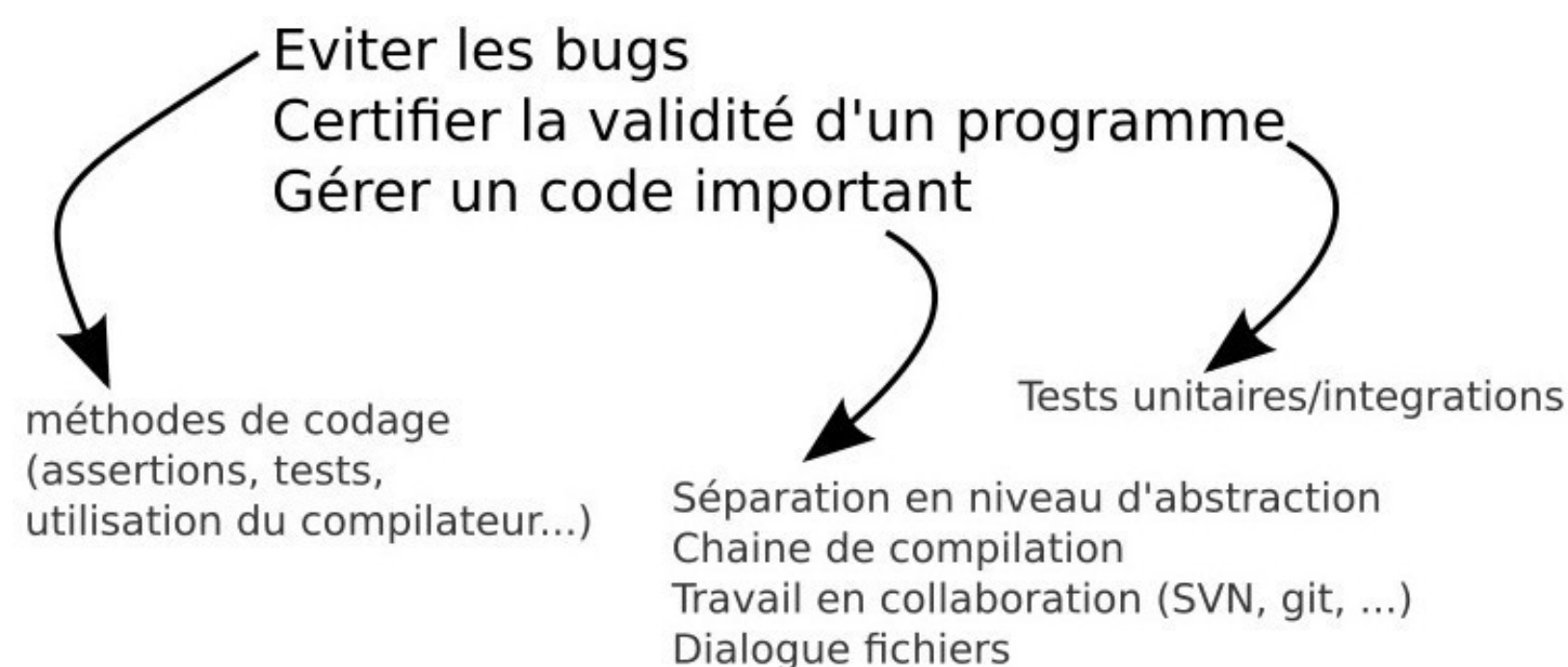
(72h)

001

But du module

Connaitre/Mettre en oeuvre:

Bonnes pratiques de codage



002

Acquis du module

Acquérir une culture générale informatique

(implicite)

language du monde informatique
licences logiciels
cycles de developpement
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
...

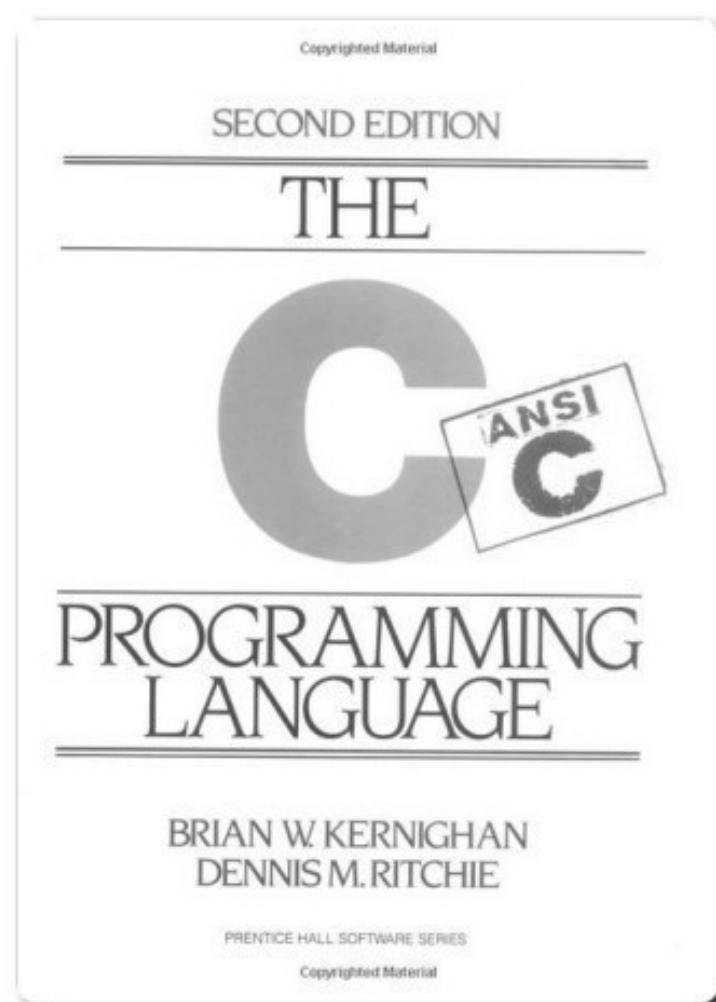
Savoir lire et s'habituer au code de vrais projets

(explicite)

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

003

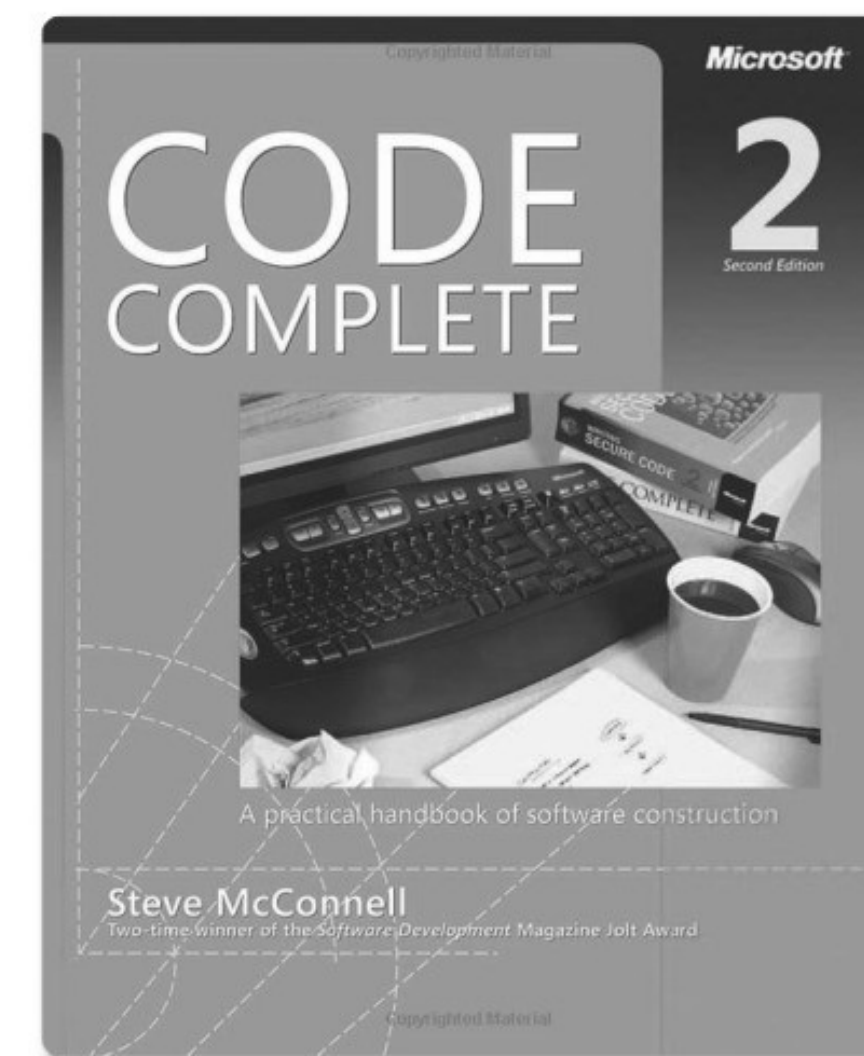
Bibliographie supplémentaire



Syntaxe C complète

004

Bibliographie supplémentaire



Bonnes pratiques de codage (générique)

005

Bibliographie supplémentaire

Les sites web:

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



+ Google

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



OpenClassrooms
<http://fr.openclassrooms.com>



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

006

Pratiquez

Informatique = pratique
= expérience

Pratiquez !

+ Pratique => meilleur emploi

recherchent ce que vous savez faire!
pas vos notes!



007

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 !

008

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*));
```

char	(1)
short	(2)
int	(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

float	(4)
double	(8)
long double	(16)

flottants

char*	(8)
int*	(8)
...	
void*	(8)

pointeurs

Sous Linux, compilé avec gcc, x86, 64bits

009

Qualité du code

Qu'est ce qu'un bon code?

010

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 ?

011

Qualité du code

Un code lisible est un code qui:

- se lit et se comprend facilement
- est sa propre documentation
- représente le plus possible des entités réelles
- s'organise en structures représentatives

Réutilisable



Peu de bugs

Maintenable

Facile à débbugger

Optimisation plus aisée

012

Qualité du code

Cas des enum

013

Qualité du code

Remarque 1

Utiliser les **enum** dès que possible

- + Expressivité d'une chaîne de caractère
- + Simplicité d'un entier



nom de l'enum valeurs possibles

```
enum type_animal {poule,lapin,cheval,chevre};
enum type_nourriture {mais,carotte,foin,herbe};

int main()
{
    enum type_animal mon_animal=lapin;
    enum type_nourriture nourriture;
    nourriture=foin;

    return 0;
}
```

automatique

affectation, manipulation comme un entier

poule : 0
lapin : 1
cheval : 2
chevre : 3

[enum/p1]

014

Enum

Exemple

```
enum type_animal {poule,lapin,cheval,chevre};
enum type_nourriture {mais,carotte,foin,herbe};

enum type_nourriture trouve_nourriture(enum type_animal animal)
{
    if(animal==poule)
        return mais;
    if(animal==lapin)
        return carotte;
    if(animal==cheval)
        return foin;
    if(animal==chevre)
        return herbe;
}

int main()
{
    enum type_animal mon_animal=lapin;
    enum type_nourriture nourriture=trouve_nourriture(mon_animal);

    printf("Nourriture a acheter: %d\n",nourriture);

    return 0;
}
```

[enum/p2]

015

Enum

Exemple, encore mieux

```
enum type_animal {poule,lapin,cheval,chevre};
enum type_nourriture {mais,carotte,foin,herbe};

enum type_nourriture trouve_nourriture(enum type_animal animal)
{
    switch(animal)
    {
        case poule:
            return mais;
        case lapin:
            return carotte;
        case cheval:
            return foin;
        case chevre:
            return herbe;
        default:
            puts("Erreur nourriture");
            abort();
    }
}

int main()
{
    enum type_animal mon_animal=lapin;
    enum type_nourriture nourriture=trouve_nourriture(mon_animal);

    printf("Nourriture a acheter: %d\n",nourriture);

    return 0;
}
```

[enum/p3]

switch/case

Gestion d'erreur

016

Enum

Avec des entiers

```
//On dit que
//poule=0, lapin=1, cheval=2, chevre=3
//mais=0, carotte=1, herbe=2, foin=3
int trouve_nourriture(int mon_animal)
{
    switch(animal)
    {
        case 0:
            return 0;
        case 1:
            return 1;
        case 2:
            return 3;
        case 3:
            return 2;
    }
}

int main()
{
    int mon_animal=2;
    int nourriture=trouve_nourriture(mon_animal);

    printf("Nourriture a acheter: %d\n",nourriture);

    return 0;
}
```

Peu compréhensible
(nécessite commentaires)
Erreurs faciles!
=> A éviter !

```
enum type_animal {poule,lapin,cheval,chevre};
enum type_nourriture {mais,carotte,foin,herbe};

enum type_nourriture trouve_nourriture(enum type_animal animal)
{
    switch(animal)
    {
        case poule:
            return mais;
        case lapin:
            return carotte;
        case cheval:
            return foin;
        case chevre:
            return herbe;
        default:
            puts("Erreur nourriture");
            abort();
    }
}

int main()
{
    enum type_animal mon_animal=lapin;
    enum type_nourriture nourriture=trouve_nourriture(mon_animal);

    printf("Nourriture a acheter: %d\n",nourriture);

    return 0;
}
```

NE PAS FAIRE

017

Enum

Avec des chaînes de caractères

```
const char* trouve_nourriture(const char* animal)
{
    if(strcmp(animal,"poule")==0)
        return "mais";
    if(strcmp(animal,"lapin")==0)
        return "carotte";
    if(strcmp(animal,"cheval")==0)
        return "foin";
    if(strcmp(animal,"chevre")==0)
        return "herbe";
}

int main()
{
    char mon_animal[50];
    strcpy(mon_animal,"lapin");
    const char* nourriture=trouve_nourriture(mon_animal);

    printf("Nourriture a acheter: %s\n",nourriture);

    return 0;
}
```

switch/case
impossible

Fonctions complexe
(strcmp,strcpy,...)
Pointeurs sous jacents
Lourd
=> A éviter

```
enum type_animal {poule,lapin,cheval,chevre};
enum type_nourriture {mais,carotte,foin,herbe};

enum type_nourriture trouve_nourriture(enum type_animal animal)
{
    switch(animal)
    {
        case poule:
            return mais;
        case lapin:
            return carotte;
        case cheval:
            return foin;
        case chevre:
            return herbe;
        default:
            puts("Erreur nourriture");
            abort();
    }
}

int main()
{
    enum type_animal mon_animal=lapin;
    enum type_nourriture nourriture=trouve_nourriture(mon_animal);

    printf("Nourriture a acheter: %d\n",nourriture);

    return 0;
}
```

NE PAS FAIRE

018

Enum

Fonction de conversion enum -> chaîne de caractères

```
enum type_animal {poule,lapin,cheval,chevre};

const char* nomme_animal(enum type_animal animal)
{
    switch(animal)
    {
        case poule:
            return "poule";
        case lapin:
            return "lapin";
        case cheval:
            return "cheval";
        case chevre:
            return "chevre";
        default:
            return "inconnu";
    }
}

int main()
{
    printf("Je suis une %s\n",nomme_animal(poule));

    return 0;
}
```

Simplicité enum
+
Expressivité chaîne

A FAIRE!

019

Enum

Les enum peuvent indexer des tableaux

```
enum nom_pays {France,
               Allemagne,
               Angleterre,
               Espagne,
               Chine};

int main()
{
    int population[6];

    population[France]=66;
    population[Allemagne]=82;
    population[Angleterre]=53;
    population[Espagne]=47;
    population[Chine]=1351;

    return 0;
}
```

Simple
Expressif

[enum/p4]



020

Qualité du code

Cas des structs

021

Cas des structs

Utilisez dès que possible
les structs pour structurer votre code

```
struct employe
{
    char nom[20];
    int age;
    int salaire;
    int anciennete;
};

struct entreprise
{
    char nom[20];
    long int chiffre_affaire;
    int benefice;
}
```

segmente en entités

représente un
objet/personnage réel

struct = 1 niveau
d'abstraction



022

Struct

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

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 du mot clé struct

023

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

[struct/p1]

024

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.prix=8500;

    //prix de la 3eme roue
    ma_bmw.roue[2].prix=95;

    //prix de la 1ere vitre
    ma_bmw.vitre[0].prix=150;

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

    return 0;
}
```

[struct/p1]

025

Struct (adresse élément)

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

int main()
{
    struct voiture ma_bmw;

    //affectation de la 3eme roue
    roue_affecte_michelin( &(ma_bmw.roue[2]) );

    return 0;
}
```

adresse de la roue

[struct/p2]

026

Struct anonyme

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

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

    return 0;
}
```

permet d'éviter la répétition du mot struct

027

Qualité du code

Cas des tableaux

028

Cas des tableaux

Ne jamais utiliser l'arithmétique
pointeur pour indexer les tableaux

```
int T[5];  
T[2]=8;
```

Lisible

```
int T[5];  
*(T+2)=8;
```

Peu lisible

```
int T[5][8];  
T[2][3]=8;
```

Lisible

```
int T[5][8];  
*(*(T+2)+3)=8;
```

Illisible

$*(p+k)$ = Exercice scolaire != Développement logiciel

029

Syntaxe pointeur/tableau

Tableau

```
int main()  
{  
    int tableau[TAILLE];  
  
    //accès à 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;  
}
```

030

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

031

Syntaxe pointeur/tableau

Bonne pratique:

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

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

Banissez: pointeurs multiples ($\text{int } **p$) ✗

structurez en abstraction ✓

032

Syntaxe pointeur/tableau

Exemples de portabilité (différents langages):

Vecteur en C++

```
int main()
{
    vector<int> mon_vecteur;
    mon_vecteur.resize(50);
    mon_vecteur[10]=5;
}
```

```
int main()
{
    vector<int> mon_vecteur;
    mon_vecteur.resize(50);
    *(mon_vecteur+10)=5;
}
```

Tableau en Python

```
tableau = [4,5,6,7];
tableau[2]=8;
```

```
tableau = [4,5,6,7];
*(tableau+2)=8;
```

Tableau en Java

```
public class Programme {
    public static void main(String[] args) {
        int tableau[] = new int [10];
        tableau[2]=8;
    }
}
```

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

033

Syntaxe pointeur/tableau

Ex. Listing de notes

Mathieu: {12,13,14,11,9,12}
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}};

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

[tableaux/p1]

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}};

    //accès 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;
}
```

[tableaux/p2]

Ecriture tableau
OK, avec documentation

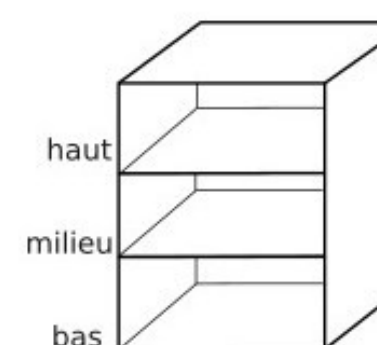
034

Syntaxe pointeur/tableau



Ex. Etagère de CD/DVD/Livres

rock blues classique techno
action horreur scifi thriller
fantaisie biographie roman cuisine



=> Enregistrement du nom de l'auteur/chanteur

Formalisme pointeur
(proche de l'encodage mémoire) ✗

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

    //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");
}
```

[tableaux/p3]

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

035

Qualité du code

Remarques sur l'optimisation

036

+ Court != + Lisible

Exemple

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++);
}
```

commentaires
indispensables
bug?

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];
    }
}
```

vraiment besoin
de commentaires?

037

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;
}
```

038

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

1

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

2

```
void fonction(int v1[],int v2[],int v3[],int N)
{
    int *p_v1=v1,*p_v2=v2,*p_v3=v3;
    register unsigned int k=0;
    while(k<N)
    {
        *p_v1_2++ += *p_v2++;
        *p_v3++ += *p_v2++;
        ++k;
    }
}
```

3

```
void fonction(int v1[],int v2[],int v3[],int N)
{
    v1[0]=v1[0]+v2[0];v3[0]=v2[0];
    v1[1]=v1[1]+v2[1];v3[1]=v2[1];
    v1[2]=v1[2]+v2[2];v3[2]=v2[2];
    v1[3]=v1[3]+v2[3];v3[3]=v2[3];
    v1[4]=v1[4]+v2[4];v3[4]=v2[4];
    v1[5]=v1[5]+v2[5];v3[5]=v2[5];
    v1[6]=v1[6]+v2[6];v3[6]=v2[6];
    v1[7]=v1[7]+v2[7];v3[7]=v2[7];
    v1[8]=v1[8]+v2[8];v3[8]=v2[8];
    v1[9]=v1[9]+v2[9];v3[9]=v2[9];
    v1[10]=v1[10]+v2[10];v3[10]=v2[10];
    v1[11]=v1[11]+v2[11];v3[11]=v2[11];
    v1[12]=v1[12]+v2[12];v3[12]=v2[12];
    v1[13]=v1[13]+v2[13];v3[13]=v2[13];
    v1[14]=v1[14]+v2[14];v3[14]=v2[14];
    v1[15]=v1[15]+v2[15];v3[15]=v2[15];
}
```

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];

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

[tableaux/p5]

039

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

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]++;
        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_v2=v2+1;
    int *p_v3=v3+1;
    register unsigned int k=1;
    while(k<N-1)
    {
        *p_v1_2++;
        *p_v3 = *p_v1++-*p_v2++;
        (*p_v3)/=10;
        ++k;
    }
}
```

3 245ms

```
void fonction(int v1[],int v2[],int v3[],int N)
{
    v1[1]++;
    v3[1]=(v1[0]-v2[2]);v3[1]/=10;
    v1[2]++;
    v3[2]=(v1[1]-v2[3]);v3[2]/=10;
    v1[3]++;
    v3[3]=(v1[2]-v2[4]);v3[3]/=10;
    v1[4]++;
    v3[4]=(v1[3]-v2[5]);v3[4]/=10;
    v1[5]++;
    v3[5]=(v1[4]-v2[6]);v3[5]/=10;
    v1[6]++;
    v3[6]=(v1[5]-v2[7]);v3[6]/=10;
    v1[7]++;
    v3[7]=(v1[6]-v2[8]);v3[7]/=10;
    v1[8]++;
    v3[8]=(v1[7]-v2[9]);v3[8]/=10;
    v1[9]++;
    v3[9]=(v1[8]-v2[10]);v3[9]/=10;
    v1[10]++;
    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];

    v3[1]++;
    v3[1]=(v1_00-v2_02);v3[1]/=10;
    v3[2]++;
    v3[2]=(v1_01-v2_03);v3[2]/=10;
    v3[3]++;
    v3[3]=(v1_02-v2_04);v3[3]/=10;
    v3[4]++;
    v3[4]=(v1_03-v2_05);v3[4]/=10;
    v3[5]++;
    v3[5]=(v1_04-v2_06);v3[5]/=10;
}
```

[tableaux/p5]

040

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

Pour 75 000 000 executions :



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

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

041

Bonnes pratiques

Remarque optimisation:

On optimise pas le code !

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

On optimise l'algorithme

(années 80)

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)

042

043

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

044

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)
    paddq   -40(%rsp), %xmm0
    movdqa  %xmm0, %xmm1
    movl    $1, -24(%rsp)
    psrldq  $8, %xmm1
    paddq   %xmm1, %xmm0
    movdqa  %xmm0, %xmm1
    psrldq  $4, %xmm1
    paddq   %xmm1, %xmm0
    movd    %xmm0, -60(%rsp)
    movl    -60(%rsp), %esi
    addl    $1, %esi
    jmp     printf
.cfi_endproc
```

\$ gcc -O2

045

Gestion code important

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

046

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;
}
```

nom_constructeur

pression

prix

roue_voiture

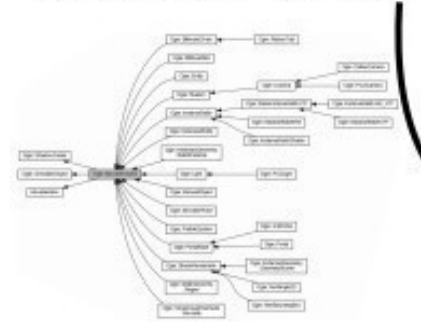
IDE Linux:
QtCreator
Eclipse
Code::Blocks

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

047

Developpement logiciel

Réaliser le **design** d'un **code volumineux**.



(reflection en amont de coder
coder
reflection en aval



(1 personne: ~30 000 lignes
équipe: 200 000 lignes
logiciel: 1 000 000 lignes

ex. Windows XP (40 000 000 lignes)

048

Gestion code important

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

049

Fichiers en-tête

Signature/en tête d'une fonction

Implémentation / corps d'une fonction

050

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;
}
```

[entete/p1]

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;
}
```

[entete/p2]

051

Fichiers en-tête

Solution 3:



052

Fichiers en-tête

Synthèse:

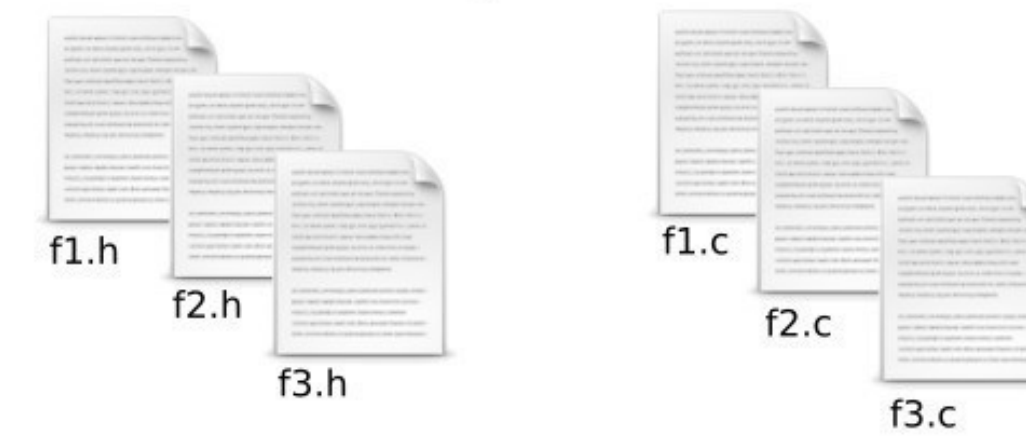
- + Evite les longs fichiers
- + Séparation en-tête / implémentation

publique
entree/sortie
stable

privée
peut changer

En pratique:

Autant de fichiers que d'abstractions



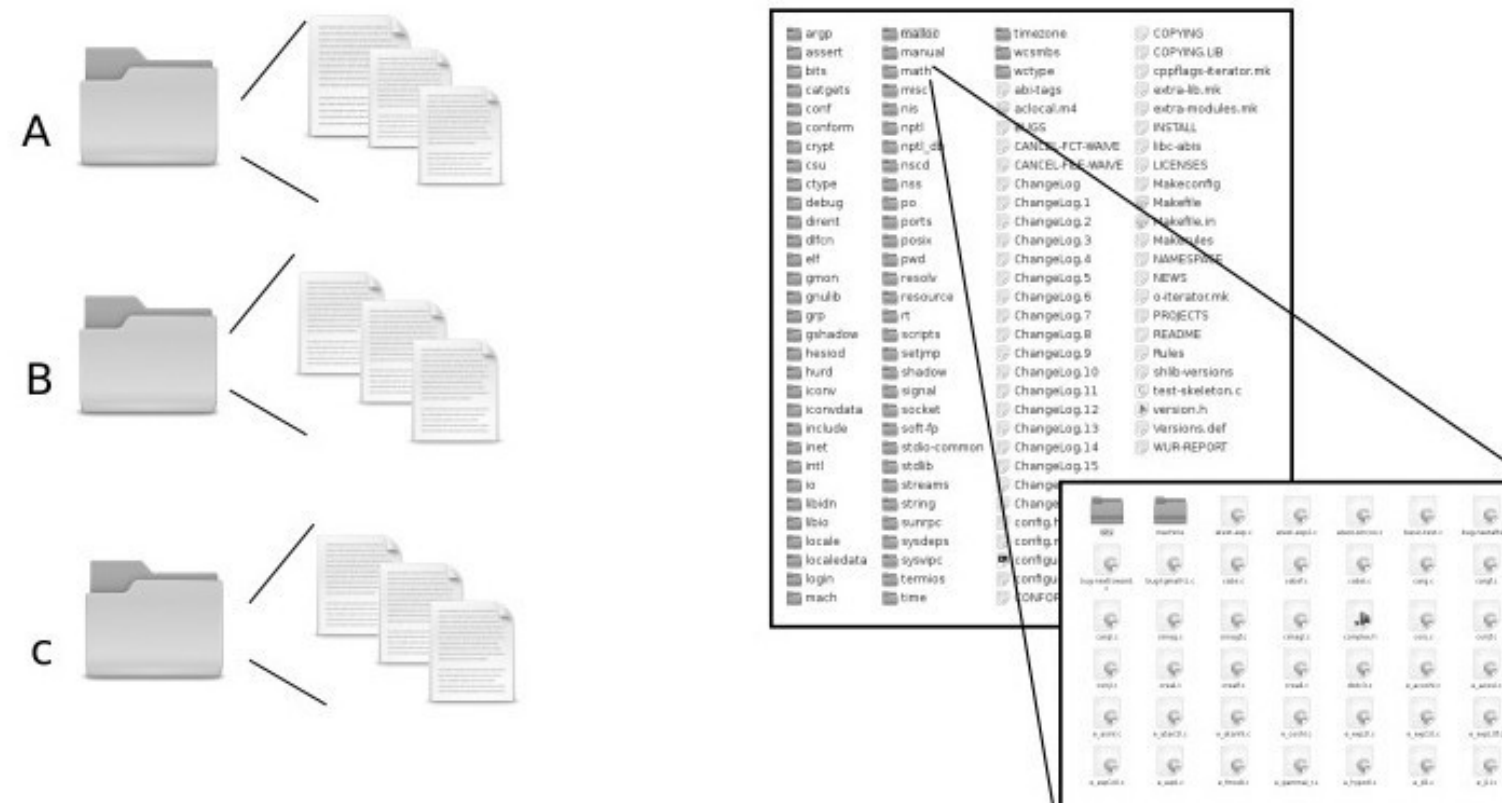
053

Fichiers en-tête

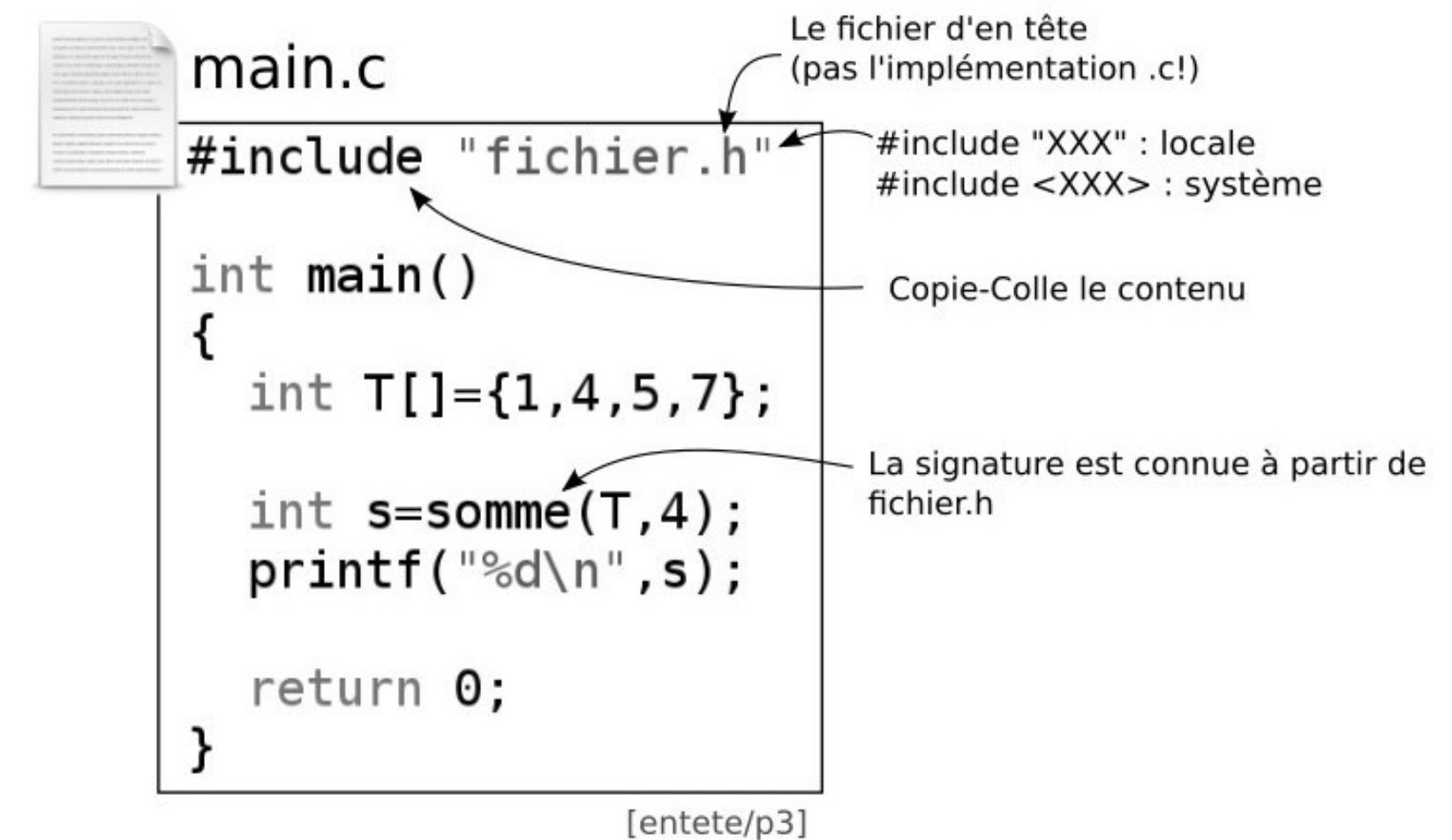
Synthèse:

Pour les très gros projets:

ex. LibC



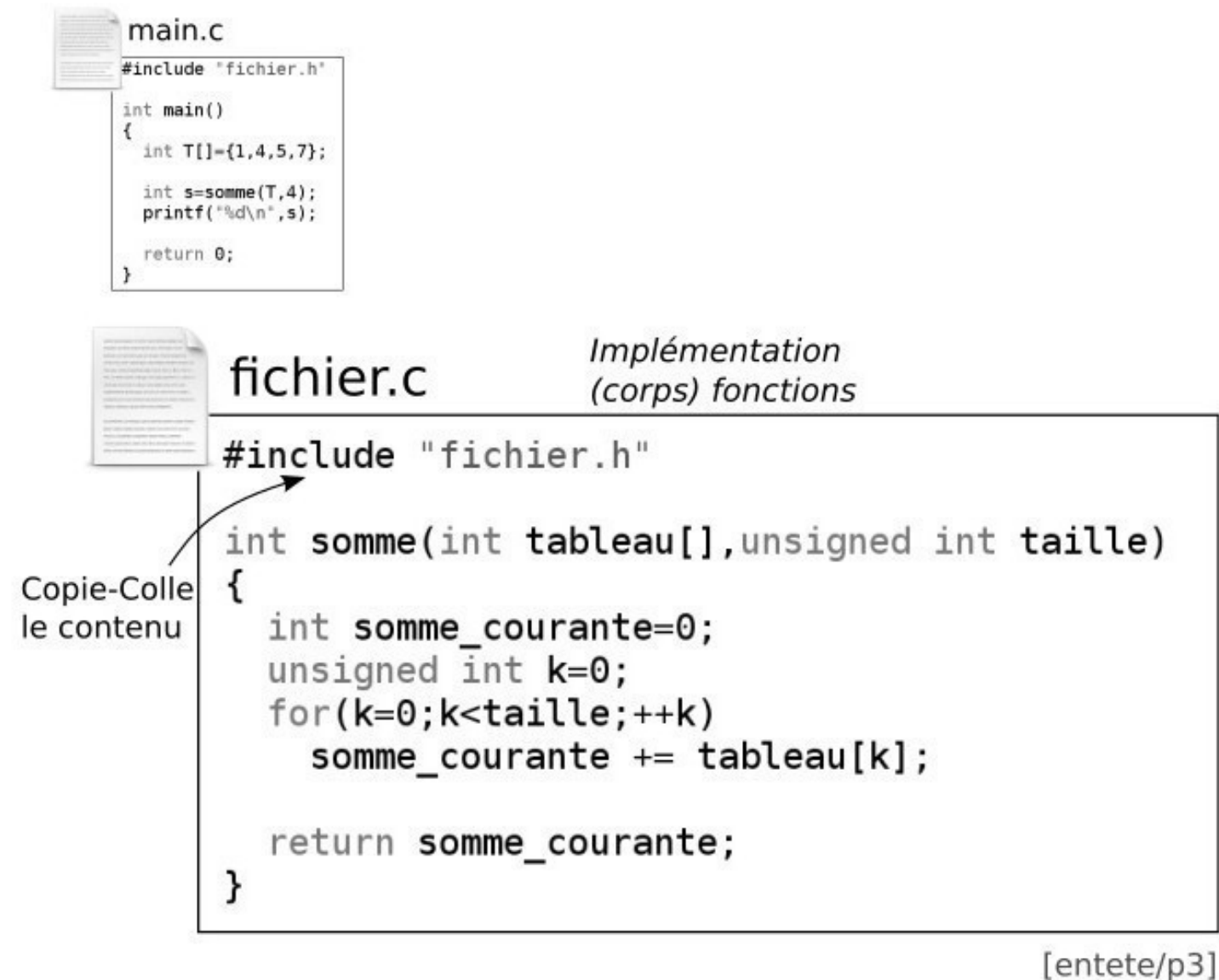
Fichiers en-tête : en pratique



054

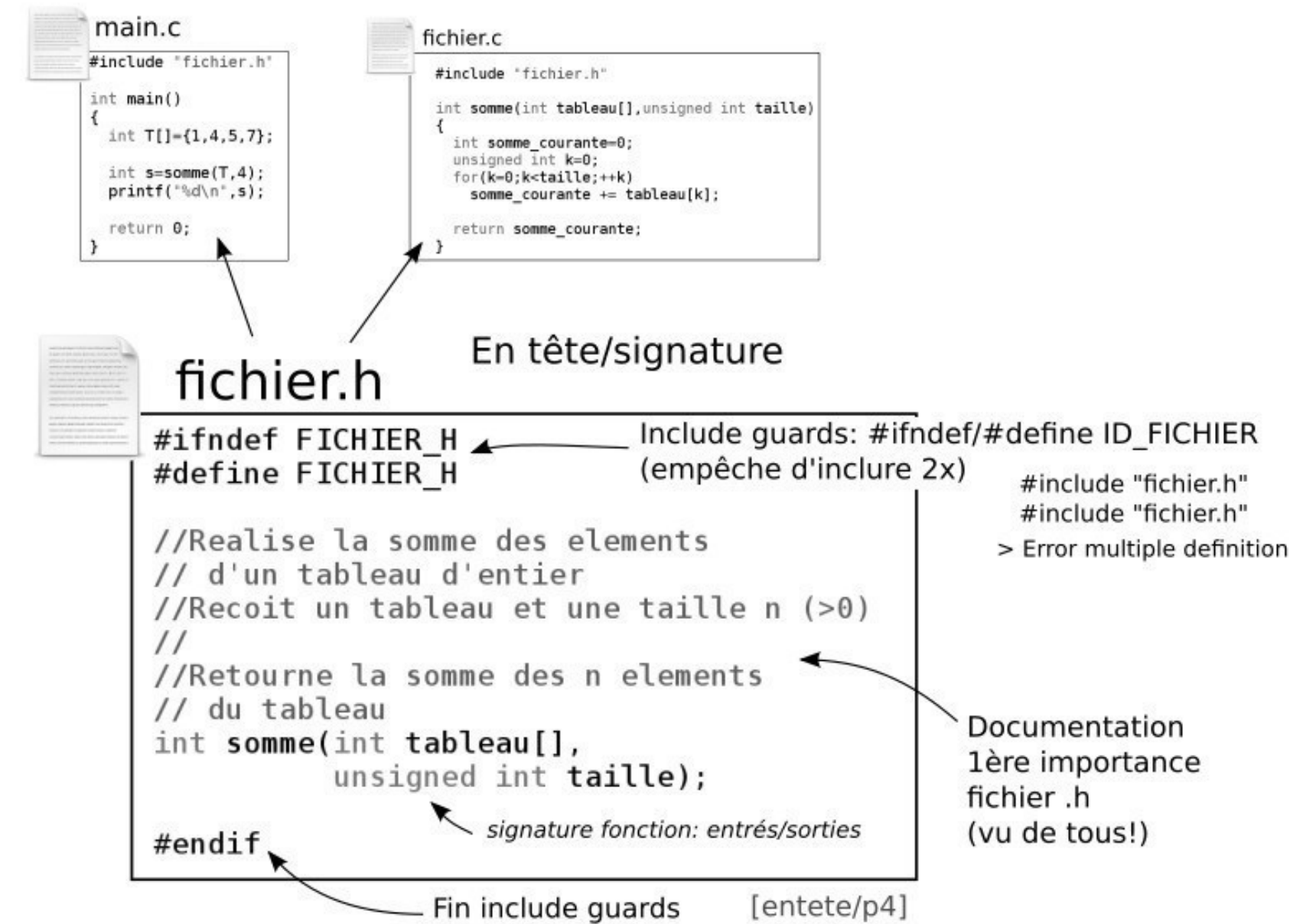
055

Fichiers en-tête : en pratique



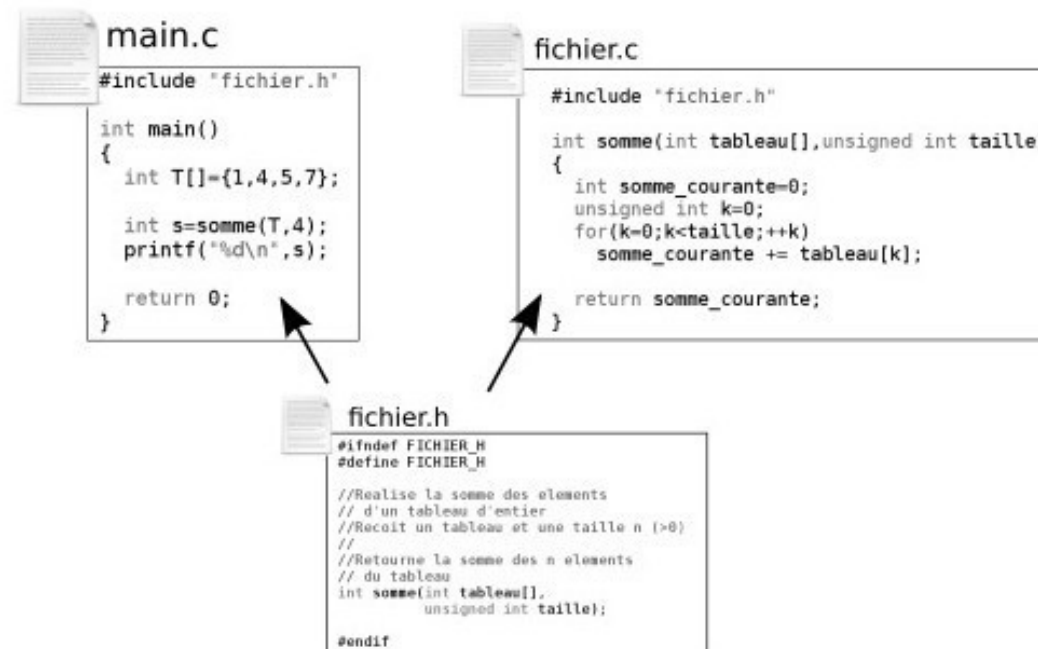
056

Fichiers en-tête : en pratique



057

Fichiers en-tête : en pratique



Compilation séparée:

```
$ gcc -c main.c -g -Wall -Wextra → Compilation main.c -> main.o
$ gcc -c fichier.c -g -Wall -Wextra → Compilation fichier.c -> fichier.o

$ gcc main.o fichier.o -o mon_executable → Edition des liens
                                         création executable
                                         avec main.o et fichier.o
```

On ne compile pas les fichier .h (ils sont copiés-collés)

058

Gestion code important

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

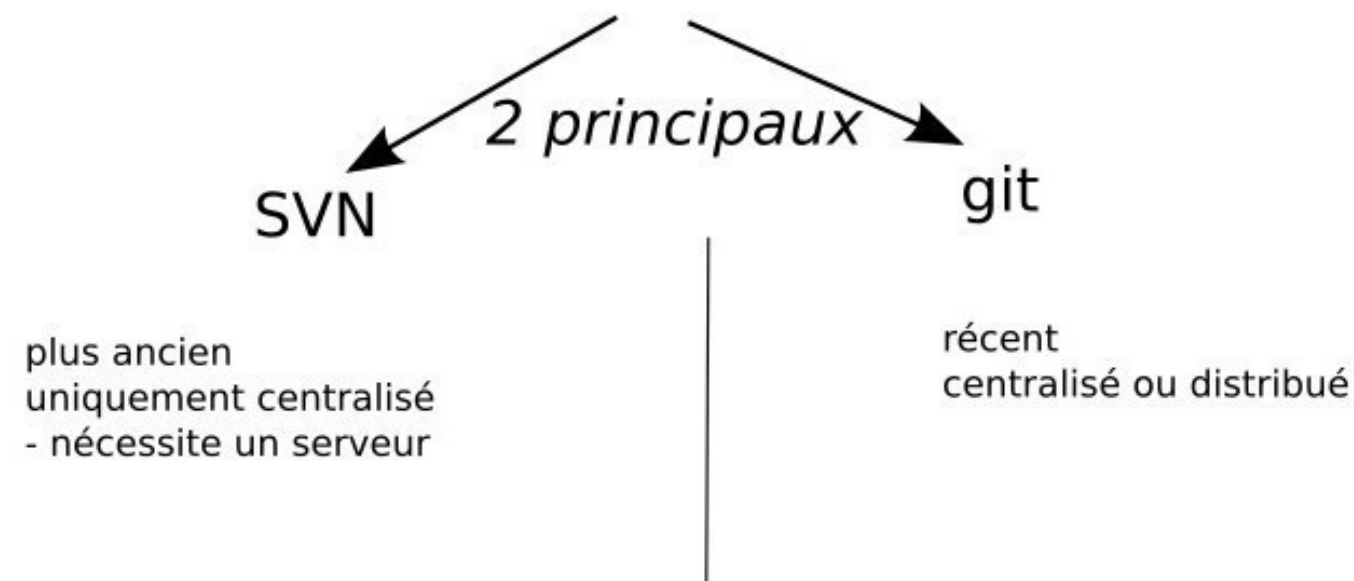
059

Programmer à plusieurs

Parties spécifiques *inter-dépendance*

Passage par mails *petit code uniquement*

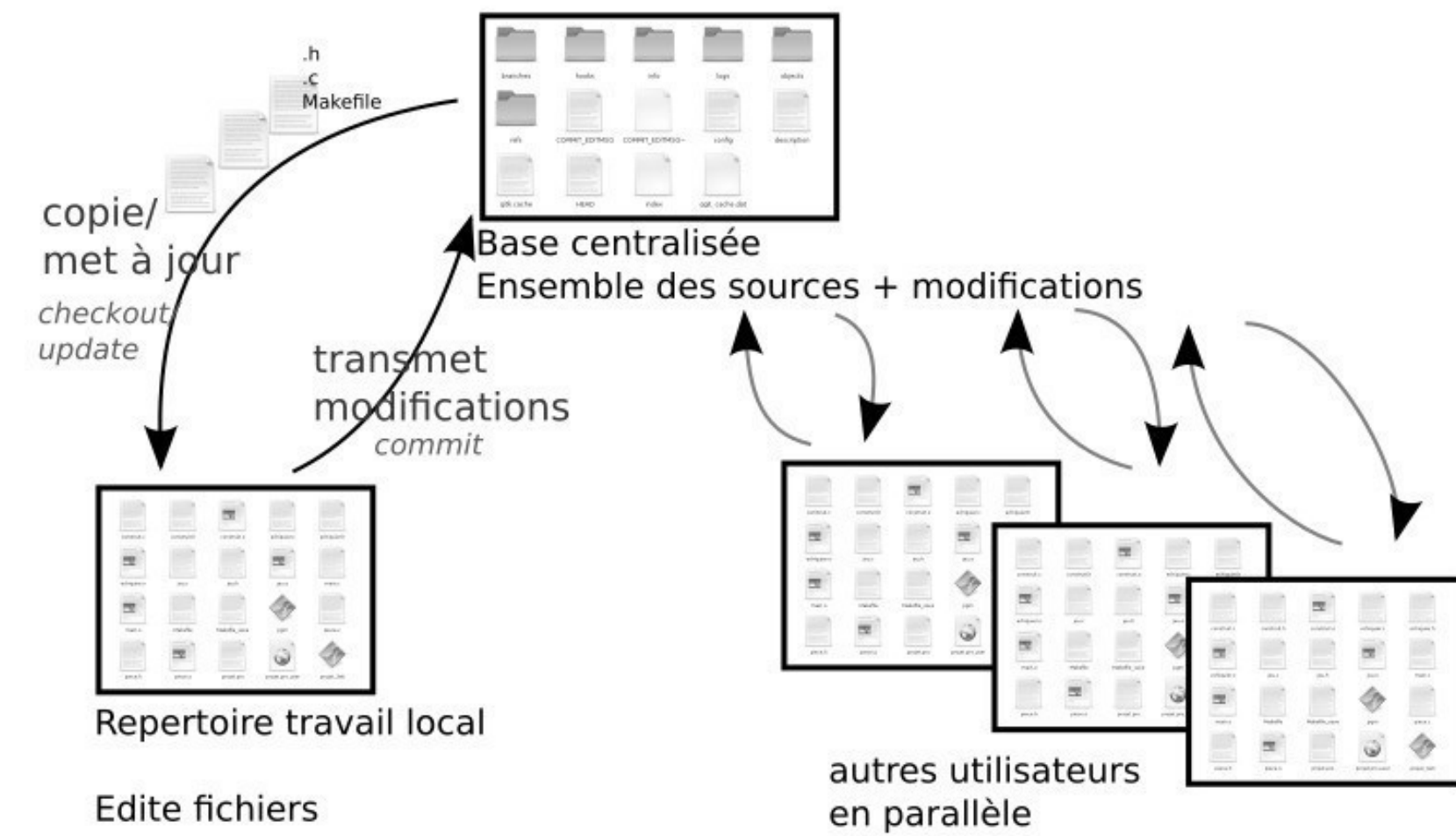
Logiciel de controle de version (pro)



060

Programmer à plusieurs

Principe d'un logiciel de controle de version



061

Programmer à plusieurs

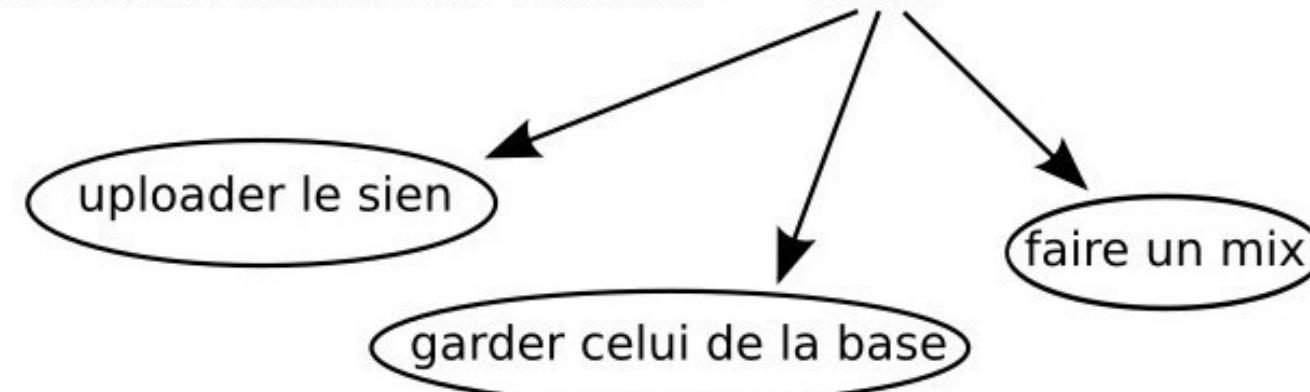
Principe d'un logiciel de controle de version

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



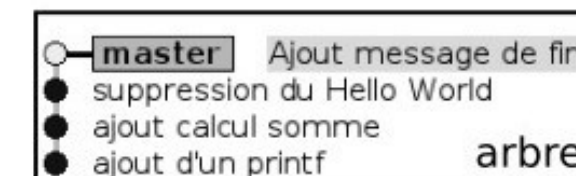
062

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

3

```
@@ -11,5 +11,7 @@ int main()
{
    printf("Hello world\n");
    int a=3;
    int b=8;
    printf("%d\n",calcul_somme(a,b));
    printf("Fin\n");
    return 0;
}
```

4

063

Programmer à plusieurs

Principe d'un logiciel de contrôle 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 suivi 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



064

Licences logiciels

065

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?

066

Logiciel Libre

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



créateur de la FSF

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, ...

067

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>



068

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

LGPL
BSD
MIT

Utilisation code LGPL => n'implique pas que l'ensemble soit LGPL
Peut être utilisé pour 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/>.
```

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

069

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

070

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
...

071

Droits d'auteurs

Droit morale	Droit patrimoniale
Paternité Choix de diffusion ...	Privilège d'exploitation (royalties, ...)
Inaliénable, Imprescriptible	
=> Il est interdit (et impossible) de changer le nom de l'auteur original!!	Note: En France pas de brevets logiciels!