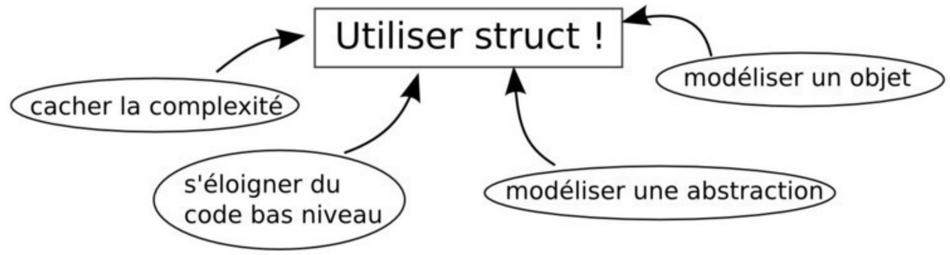
Code de haut niveau:

Modélisation d'abstraction

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

Intérêt des structs



+ fonctions manipulants struct

Intéret structs: Lisibilité

Préférer:

```c
struct vecteur
{
    float x;
    float y;
};

float norme(const struct 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);
    return 0;
}
```

lisible
on connait quelles
règles suit l'objet

Plutôt que:

```c
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

[haut_niveau/p1]

Intéret structs: Lisibilité

ex. Généricité

```c
int main()                                  OK
{
    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);
}
```

```c
int main()                                  A NE PAS FAIRE
{
    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]);
}
```

=> Changez la norme 2 vers une norme infinie

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

...

1 ligne modifée, 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;
};
```

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 faire

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

√

- A ne pas faire

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

X

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

X

```
#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;
};
```

caractéristiques haut niveau

descripteur de fichier niveau système:

Mélange:
détail d'implémentation
/caractéristiques haut niveau

╳

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 donnees
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
```

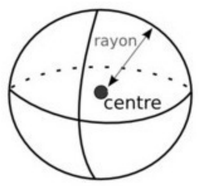
```
void sphere_init(struct sphere* s,float centre_x,
                                  float centre_y,
                                  float centre_z,
                                  float rayon);

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);
}
```

[haut_niveau/p2]

Struct: Notion d'abstraction

Implémentation 1:

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



implémentation
fonctions

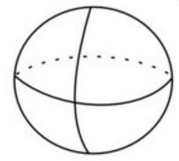
```
void sphere_init(struct sphere* s,float centre_x,
                                  float centre_y,
                                  float centre_z,
                                  float rayon)
{
    assert(rayon>0);

    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);
}
```

[haut_niveau/p2]

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

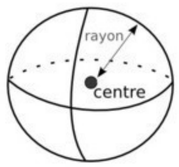
implémentation
fonctions

```
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);
}
```

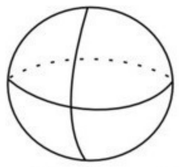
[haut_niveau/p3]

Struct: Notion d'abstraction

Encapsulation:



VS



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

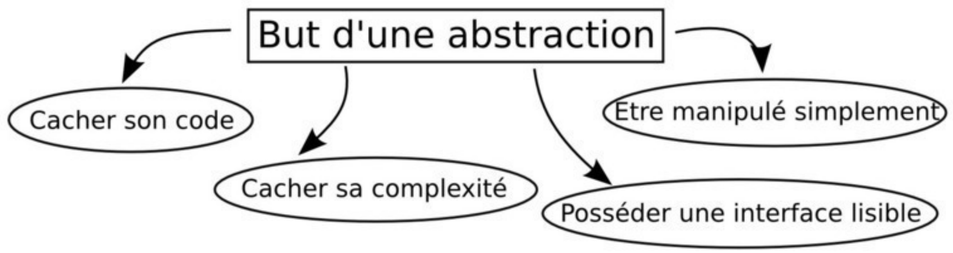
Peu importe l'implémentation
=> **L'utilisation est la même**

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

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
+ evolutif

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

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:

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

//charactere
char charactere='0';

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

//tableaux statiques
int tableau_1[5]={0,0,0,0,0};
char tableau_2[5]={'\0','\0','\0','\0','\0'};
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;
```

généralement
entiers à 0 (ou -1)

généralement
flottants à 0

\0: fin de chaine

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

Initialisation des Variables

Exemple:

```c
int entier;
printf("%d\n",entier);

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

```
32767
379680036110150551298113536.000000
```

Dépend du:

système
compilateur
code
état de la mémoire (autres programmes)

=> **comportement indeterminé**

Initialisation des Structs

Les structs doivent avoir leur fonction d'initialisation

Exemple:

```c
#define TAILLE_NOM_MAX 60

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

Pas d'initialisation: ✗

```c
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:

```c
#define TAILLE_NOM_MAX 60

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

Fonction d'initialisation: ✓

```c
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;
}
```

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

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

[haut_niveau/p4]

Initialisation des Structs

Exemple de cas d'erreur:

```c
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: [haut_niveau/p5]

```c
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;
}
```


Initialisation des Structs

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

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
```

$ gcc -O2

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

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

sans initialisation

Il n'y a aucune gain de perfomance 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 ?

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);
}
```

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);
}
```

$ gcc -O2

Assembleur identique

```
main:
.LFB3:
        .cfi_startproc
        subq    $24, %rsp
        .cfi_def_cfa_offset 32
        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
```

=> 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!

Initialisation des Structs

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'entitée 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 longeur();
```

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,undescore, ...)

```
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_eliminatoire=3;
int note_passage=10;

int recoit_note(int note_etudiant,int id_etudiant)
{
    if(note_etudiant>note_eliminatoire)
    {
        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.

✓

```c
#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";


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

void charge_nom_etudiants(struct classe* c,const char* filename);
void envoie_mail(const char* titre);
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

X

peu compréhensible

```c
#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";


int classe_recupere_note(const struct c* c,
                         const char* n);
void charge_nom_etudiants(struct c* c,const char* filename);
void envoie_mail(const char* titre);

int main()
{
  struct c 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 c* 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[indice_de_parcours].n,
            nom_de_l_etudiant);
      if(retour_comparaison_chaine_caractere==0)
    return classe_etudiant_courante->classe[indice_de_parcours].x;
      ++indice_de_parcours;
    }
  printf("Etudiant %s non trouve\n",nom_de_l_etudiant);
  return -1;
}
```

inutillement complexe
peu lisible