

# Projet de Programmation

Benoit Donnet  
Année Académique 2023 - 2024



1

## Agenda

### **Partie 1: Techniques Avancées de C**

- Chapitre 1: Définition de Type
- Chapitre 2: Généricité
- Chapitre 3: Arguments d'un Programme

# Agenda

- Chapitre 1: Définition de Type
  - Modélisation des Données
  - Include Guards
  - Type Opaque

# Agenda

- Chapitre 1: Définition de Type
  - Modélisation des Données
    - ✓ Enumération
    - ✓ Intérêt des Enregistrements
    - ✓ Bonnes/Mauvaises Pratiques
  - Include Guards
  - Type Opaque

# Énumération

- Rappel
  - une énumération permet de définir des noms symboliques correspondant à des valeurs entières
  - cfr. INFO0946, Chap. 5
- Il faut utiliser des énumérations dès que possible
  - simplicité d'un entier
  - plus expressif qu'une chaîne de caractères

## Énumération (2)

- Exemple

```
typedef enum{poule, lapin, cheval, chevre} TypeAnimal;
typedef enum{mais, carotte, foin, herbe} TypeNourriture;

int main(){
    TypeAnimal monAnimal = lapin;
    TypeNourriture nourriture;

    nourriture = foin;

    return 0;
} //fin programme
```

# Enumération (3)

```
TypeNourriture trouve_nourriture(TypeAnimal animal){
    switch(animal){
        case poule: return mais;
        case lapin: return carotte;
        case cheval: return foin;
        case chevre: return herbe;
        default:
            printf("Erreur nourriture!\n");
            abort();
    } //fin switch()
} //fin trouve_nourriture()

int main(){
    TypeAnimal monAnimal = lapin;
    TypeNourriture nourriture = trouve_nourriture(monAnimal);
    printf("Nourriture à acheter: %d\n", nourriture);

    return 0;
} //fin programme
```

# Enumération (4)

```
char *animal_vers_chaine(TypeAnimal animal){
    switch(animal){
        case poule: return "poule";
        case lapin: return "lapin";
        case cheval: return "cheval";
        case chevre: return "chevre";
        default: return "inconnu";
    } //fin switch()
} //fin animal_vers_chaine()

int main(){
    printf("je suis un: %s\n", animal_vers_chaine(cheval));

    return 0;
} //fin programme
```

# Intérêt Enregistrements

- Pourquoi définir des enregistrements?
  - cacher la complexité
    - ✓ cfr. Type Opaque
  - s'éloigner du code de bas niveau
    - ✓ cacher les détails d'implémentation
  - modéliser une abstraction
  - modéliser un objet
- On peut très facilement écrire des modules manipulant des enregistrements

## Intérêt Enregistrements (2)

- Un enregistrement sera toujours plus lisible

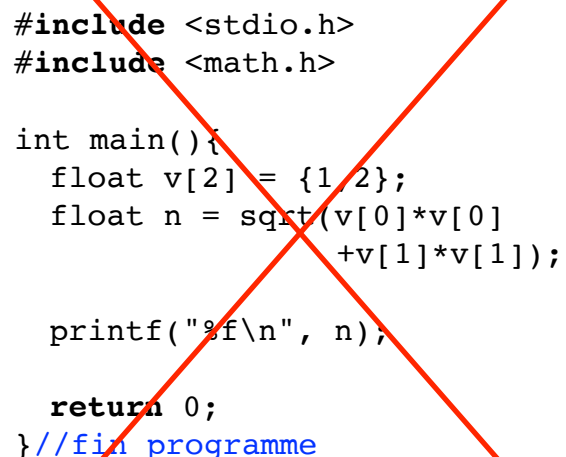
```
#include <stdio.h>
#include <math.h>

typedef struct{
    float x, y;
}Vecteur;

float norme(Vecteur *v){
    return sqrt(v->x*v->x
                +v->y*v->y);
} //fin norme()

int main(){
    Vecteur v={1,2};
    printf("%f\n", norme(&v));

    return 0;
} //fin programme
```



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

int main(){
    float v[2] = {1,2};
    float n = sqrt(v[0]*v[0]
                  +v[1]*v[1]);

    printf("%f\n", n);

    return 0;
} //fin programme
```

# Intérêt Enregistrements (3)

- L'intérêt d'un enregistrement est de cacher l'implémentation
- Il faut mettre en avant l'objet (abstrait) manipulé plutôt que les éléments qui le composent

# Bonnes/Mauvaise Pratiques

- Un *bon* enregistrement
  - encapsule des données
  - modélise un objet/une abstraction
  - contient des paramètres homogènes
- Un *mauvais* enregistrement
  - n'apporte pas d'information
  - contient des informations sans cohérence, ne modélise rien
  - ne sert que de conteneur de variables au niveau C
  - contient des noms peu significatifs
    - ✓ cfr. Coding Style

# Bonnes/Mauvaise Pratiques (2)

- Bien
  - bonne dénomination
  - haut niveau

```
typedef enum{diesel, essence, lpg} TypeCarburant;  
typedef struct{  
    int kilometrage;  
    int immatriculation;  
    TypeCarburant carburant;  
}Voiture;  
  
int main(){  
    Voiture v1;  
    v1.carburant = diesel;  
  
    return 0;  
}//fin programme
```

# Bonnes/Mauvaise Pratiques (3)

- Pas bien
  - trop proche du code
  - niveau d'abstraction non homogène
    - ✓ mélange int/voiture

```
typedef struct{  
    int unbr; //immatriculation  
    int ukm; //kilometrage  
    int uc; //carburant (0:diesel, 1:essence, 2:lpg)  
}Triplet;  
  
int main(){  
    Triplet v2;  
    v.uc = 2;  
  
    return 0;  
}//fin programme
```

# Bonnes/Mauvaise Pratiques (4)

- Bien
  - homogène
  - modulaire

```
#define N 50
typedef struct{
    int hauteur;
    char couleur[N];
    int largeur;
}TroncArbre;

typedef struct{
    int nombre;
    char couleur[N];
}FeuilleArbre;

typedef struct{
    int profondeur;
}RacineArbre;
```

```
typedef struct{
    TroncArbre tronc;
    FeuilleArbre feuille;
    RacineArbre racine;
}Arbre;
```

# Bonnes/Mauvaise Pratiques (5)

- Pas bien
  - absence d'homogénéité
  - non modulaire

```
#define N 50
typedef struct{
    int hauteurTronc;
    char couleurTronc[N];
    int largeurTronc;
    int nombreFeuille;
    char couleurFeuille[N];
    int profondeurRacine;
}Arbre;
```



# Bonnes/Mauvaise Pratiques (6)

- Pas bien
  - trop de modularité tue la modularité

```
#define N 50

typedef struct{int n;}Nombre;
typedef struct{Nombre h;}Hauteur;
typedef struct{Nombre l;}Largeur
typedef struct{char valeur[N];}Nom;
typedef struct{Nom c;}Couleur;

typedef struct{
    Hauteur h;
    Largeur l;
    Couleur c;
}TroncArbre;
```

# Bonnes/Mauvaise Pratiques (7)

- Pas bien
  - niveau d'abstraction "inhomogène"
  - mélange de caractéristiques de haut niveau et de niveau système

```
typedef enum{
    epicea, chataignier, chene, eucalyptus
}TypeArbre;

typedef struct{
    int largeur;
    int hauteur;
    TypeArbre type;
    FILE *fid_disque;
}Arbre;
```

caractéristiques de haut niveau

caractéristique niveau système

# Bonnes/Mauvaise Pratiques (8)

- Pas bien
  - trop de champs
  - illisible
  - difficile à retenir

```
typedef struct{  
    int largeur, hauteur;  
    TypeArbre type;  
    int nombreEmbranchement, circonference;  
    int poids, profondeurSousSol;  
    int nombreFleurs, fluxSeve;  
    int moisFleurissement, nombreJourFleurissement;  
    int temperatureMaximale, quantiteEau;  
    int valeurMarche, resistanceVent;  
}Arbre;
```

## Agenda

- Chapitre 1: Définition de Type
  - Modélisation des Données
  - Include Guards
    - ✓ Problème
    - ✓ Principe
    - ✓ Fonctionnement
    - ✓ Bonne Pratique
  - Type Opaque

# Problème

- Soit une application permettant de gérer la location de voiture
- Le programme est composé de
  - `voiture.{h/c}`
    - ✓ gestion de la structure de données `Voiture`
  - `trajet.{h/c}`
    - ✓ gestion d'un trajet pour une `Voiture`
  - `main-location.c`
    - ✓ programme principal

## Problème (2)

- Fichier `voiture.h`

```
typedef struct{
    int essence;
    int kilometrage;
}Voiture;
```

```
Voiture *cree_voiture(int essence, int kilometrage);
```

- Fichier `trajet.h`

```
#include "voiture.h"
```

```
void trajet_bruxelles_liege(Voiture *v);
Void trajet_mons_anvers(Voiture *v);
```

# Problème (3)

- Fichier `voiture.c`

```
#include <stdlib.h>

#include "voiture.h"

Voiture *cree_voiture(int essence, int kilometrage){
    Voiture *v = malloc(sizeof(Voiture));
    if(v==NULL)
        return NULL;

    v->essence = essence;
    v->kilometrage = kilometrage;

    return v;
} //fin cree_voiture()
```

# Problème (4)

- Fichier `trajet.c`

```
#include "trajet.h"
#include "voiture.h"

Voiture *trajet_bruxelles_liege(Voiture *v){
    v->essence -= 4;
    v->kilometrage += 97;
} //fin trajet_bruxelles_liege()

Voiture *trajet_mons_anvers(Voiture *v){
    v->essence -= 5;
    v->kilometrage += 119;
} //fin trajet_mons_anvers()
```

# Problème (5)

- Fichier main-location.c

```
#include "trajet.h"
#include "voiture.h"

int main(){
    Voiture *v = cree_voiture(45, 0);
    if (v == NULL)
        return 1;
    trajet_bruxelles_liege(v);

    return 0;
} //fin programme
```

# Problème (6)

- Compilation

```
$>gcc -o main-location voiture.c trajet.c main-location.c
```

```
In file included from trajet.c:5:
```

```
./voiture.h:4:2: error: typedef redefinition with different types  
( 'struct Voiture' vs 'struct Voiture' )
```

```
}Voiture;
```

```
^
```

```
./voiture.h:4:2: note: previous definition is here
```

```
}Voiture;
```

```
^
```

```
1 error generated.
```

```
In file included from main-location.c:2:
```

```
./voiture.h:4:2: error: typedef redefinition with different types  
( 'struct Voiture' vs 'struct Voiture' )
```

```
}Voiture;
```

```
^
```

```
./voiture.h:4:2: note: previous definition is here
```

```
}Voiture;
```

```
^
```

```
1 error generated.
```

# Problème (7)

- Problème d'inclusion multiple

- gcc -E main-location.c > toto

```
...  
# 1 "main-location.c" 2  
# 1 "./trajet.h" 1  
# 1 "./voiture.h" 1  
typedef struct{  
    int essence;  
    int kilometrage;  
}Voiture;  
Voiture *cree_voiture(int essence, int kilometrage);
```

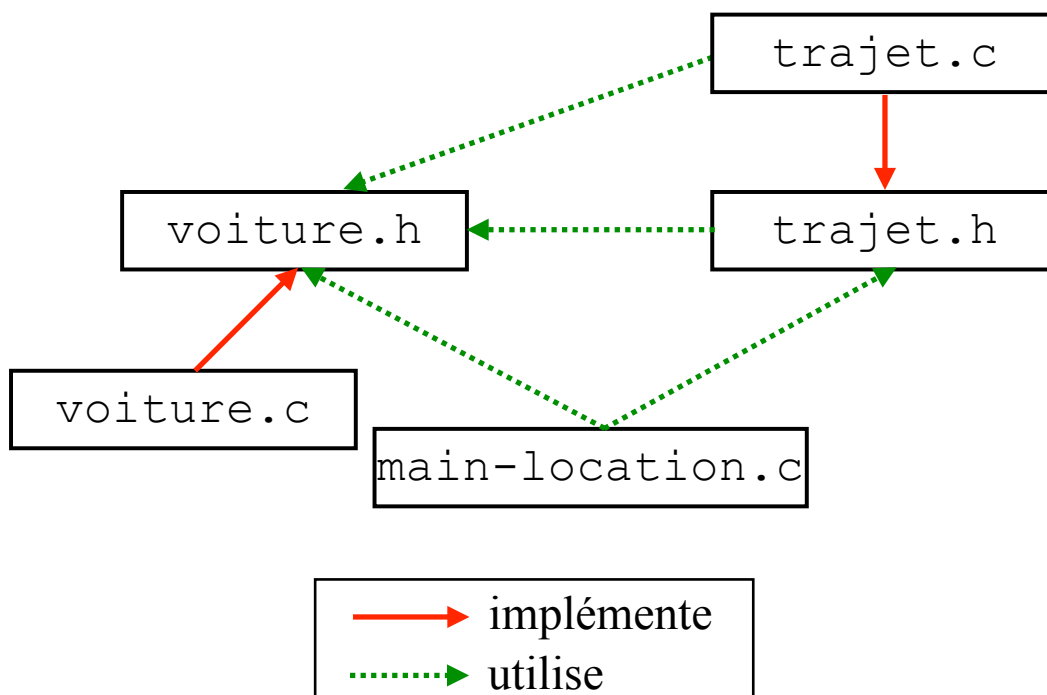
inclusion de voiture.h

```
# 2 "./trajet.h" 2  
Voiture *trajet_bruxelles_liege(Voiture *v);  
Voiture *trajet_mons_anvers(Voiture *v);  
# 2 "main-location.c" 2  
# 1 "./voiture.h" 1  
typedef struct{  
    int essence;  
    int kilometrage;  
}Voiture;  
Voiture *cree_voiture(int essence, int kilometrage);  
# 3 "main-location.c" 2  
...
```

inclusion de trajet.h

# Problème (8)

- En fait, la situation est la suivante



# Problème (9)

- Solution 1
  - n'inclure `trajet.h` que dans `main-location.c`
  - ingérable pour un grand projet
- Solution 2
  - faire en sorte que le compilateur n'inclue pas deux fois `voiture.h`
  - *include guards*
    - ✓ *garde fou*

# Principe

- **Include guard**
  - garde fou
- Construction utilisée afin d'éviter le problème de l'inclusion multiple pouvant provenir des directives de pré-processing (**#include**)
- L'ajout de include guards à un header permet de le rendre *idempotent*

# Principe (2)

- Les include guards se basent sur 2 mécanismes du pré-processeur
  - *compilation conditionnelle*
    - ✓ **#ifndef**
    - ✓ **#endif**
  - *macro*
    - ✓ **#define**

# Fonctionnement

- Fichier `voiture.h`

```
#ifndef __VOITURE__
#define __VOITURE__

typedef struct{
    int essence;
    int kilometrage;
}Voiture;

Voiture *cree_voiture(int essence, int kilometrage);

#endif
```

- On ne change rien à `trajet.h` ni à `main-location.c`



# Fonctionnement (2)

- Application de l'include guard
  - gcc -E main-location.c > toto

```
...
# 1 "main-location.c" 2
# 1 "./trajet.h" 1
# 1 "./voiture.h" 1
typedef struct{
    int essence;
    int kilometrage;
}Voiture;
Voiture *cree_voiture(int essence, int kilometrage);
# 2 "./trajet.h" 2
void trajet_bruxelles_liege(Voiture *v);
void trajet_mons_anvers(Voiture *v);
# 2 "main-location.c" 2
int main(){
    Voiture *v = cree_voiture(45, 0);
    if (v == NULL)
        return 1;
    trajet_bruxelles_liege(v);

    return 0;
}
...
```

## Bonne Pratique

- Pour tout fichier d'en-tête de nom `nom.h`
  - placer un include guard de la forme

```
#ifndef __NOM__
#define __NOM__

//contenu du header

#endif
```

- But?
  - éviter les inclusions multiples
  - éviter les collisions dues aux inclusions multiples

# Agenda

- Chapitre 1: Définition de Type
  - Modélisation des Données
  - Include Guards
  - Type Opaque
    - ✓ Principe
    - ✓ Implémentation
    - ✓ Exemple

# Principe

- **Type Opaque?**
  - structure de données incomplètement définie dans une interface
  - manipulation de la donnée à travers des fonctions/procédures
    - ✓ *accesseurs*
- **Objectifs?**
  - cacher à l'utilisateur la représentation du type
  - éviter que l'utilisateur ne modifie, à sa guise, la structure de données
- Typiquement, on définit un pointeur sur le type opaque et le manipule grâce aux fonctions/procédures
  - **pointeur opaque**
- **Exemple**
  - FILE

# Implémentation

- Comment implémenter un type opaque?
- Dans le header
  - définir incomplètement la structure
    - ✓ **forward declaration**
    - ✓ utilisation du **typedef**
  - manipuler le type via des pointeurs
    - ✓ obligatoire car le compilateur doit connaître la quantité de mémoire nécessaire associée au type
    - ✓ la taille d'un pointeur est standard
  - définir constructeur, destructeur et accesseurs
- Dans le module
  - terminer la définition de la structure

# Exemple

- Exemple
  - structure de données manipulant un point du plan
- Fichier `point.h`

```
#ifndef __POINT__  
#define __POINT__
```

```
typedef struct Point_t Point;
```

déclaration incomplète

```
//définition du constructeur/destructeur/accesseurs
```

```
#endif
```

## Exemple (2)

- Fichier `point.h` (cont.)

//définition du constructeur/destructeur/accesseurs

`Point *create_point(float x, float y);`

constructeur

`void destroy_point(Point *p);`

destructeur

`float get_x(Point *p);`

`float get_y(Point *p);`

accesseurs en lecture

`Point *set_x(Point *p, float x);`

`Point *set_y(Point *p, float y);`

accesseurs en écriture

//chaîne de caractères représentant un point

`char *to_string(Point *p);`

## Exemple (3)

- Fichier `point.c`

`#include <stdlib.h>`

`#include <assert.h>`

`#include <stdio.h>`

`#include "point.h"`

`struct Point_t{  
 float x;  
 float y;  
};`

Définition de la structure

# Exemple (4)

- Fichier point.c (cont.)

```
Point *create_point(float x, float y){
    Point *p = malloc(sizeof(Point));
    if(p==NULL)
        return NULL;

    p->x = x;
    p->y = y;

    return p;
} //end cree_point()

void destroy_point(Point *p){
    assert(p!=NULL);

    free(p);
} //end detruit_point()
```

# Exemple (5)

- Fichier point.c (cont.)

```
float get_x(Point *p){
    assert(p!=NULL);

    return p->x;
} //end get_x()

float get_y(Point *p){
    assert(p!=NULL);

    return p->y;
} //end get_y()
```

# Exemple (6)

- Fichier `point.c` (cont.)

```
Point *set_x(Point *p, float x){
    assert(p!=NULL);

    p->x = x;

    return p;
} //end set_x()

Point *set_y(Point *p, float y){
    assert(p!=NULL);

    p->y = y;

    return p;
} //end set_y()
```

# Exemple (7)

- Fichier `point.c` (cont.)

```
char *to_string(Point *p){
    assert(p!=NULL);

    char *s = malloc(sizeof(char)*32);
    if(s==NULL)
        return NULL;

    sprintf(s, "(%f, %f)", p->x, p->y);

    return s;
} //end to_string()
```

# Exemple (8)

- Fichier `test_point.c`

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

int main(){
    Point *p = create_point(3, 4);
    if (p == NULL)
        return 1;

    printf("%s\n", to_string(p));
    printf("%f\n", p->x);

    return 0;
} //end program
```

pas accepté à la compilation  
(car définition incomplète du type  
dans `point.h`)

# Exemple (9)

- Fichier `test_point.c` (cont.)

```
#include <stdio.h>
#include "point.h"
int main(){
    Point *p = create_point(3, 4);
    if (p == NULL)
        return 1;

    printf("%s\n", to_string(p));
    printf("%f\n", get_x(p));

    p = set_y(p, -5);
    printf("%s\n", to_string(p));

    destroy_point(p);
    p = NULL;
    return 0;
} //end program
```