Programmation langage C
Section 12 : Structures

Présentation de **Kevin TRANCHO**

dispensé en classe de seconde année

à l'**ESGI** Paris
(Année scolaire 2022 - 2023)

ESGI
école supérieure de
génie informatique

Introduction

Comment structurer les données pour avoir des types offrant une sémantique pertinente ?

typedef : création de synonymes

```c
/* typedef Type Synonyme; */

typedef unsigned int uint;

int main() {

  uint entierPositif = 4000000000;
  printf("%u\n", entierPositif);

  exit(EXIT_SUCCESS);
}
```

typedef : création de synonymes

```
typedef int intListeStatique[];
/* intListeStatique sera équivalent au type d'un
 ↪  tableau de int */

typedef int * intListe;
/* intListe sera équivalent au type d'un pointeur sur
 ↪  un int */
```

typedef : création de synonymes

```c
/* équivalent à avoir "int * liste" en argument */
void afficherIntListe(intListe liste) {
  int i;
  for(i = 0; liste[i] >= 0; ++i) {
    if(i) printf(", ");
    printf("%d", liste[i]);
  }
  printf("\n");
}
```

typedef : création de synonymes

```c
int main() {
  /* équivalent à "int liste[] = {1, 2, 3, 4, -1};"
  ↪  */
  intListeStatique liste = {1, 2, 3, 4, -1};
  afficherIntListe(liste);

  exit(EXIT_SUCCESS);
}
```

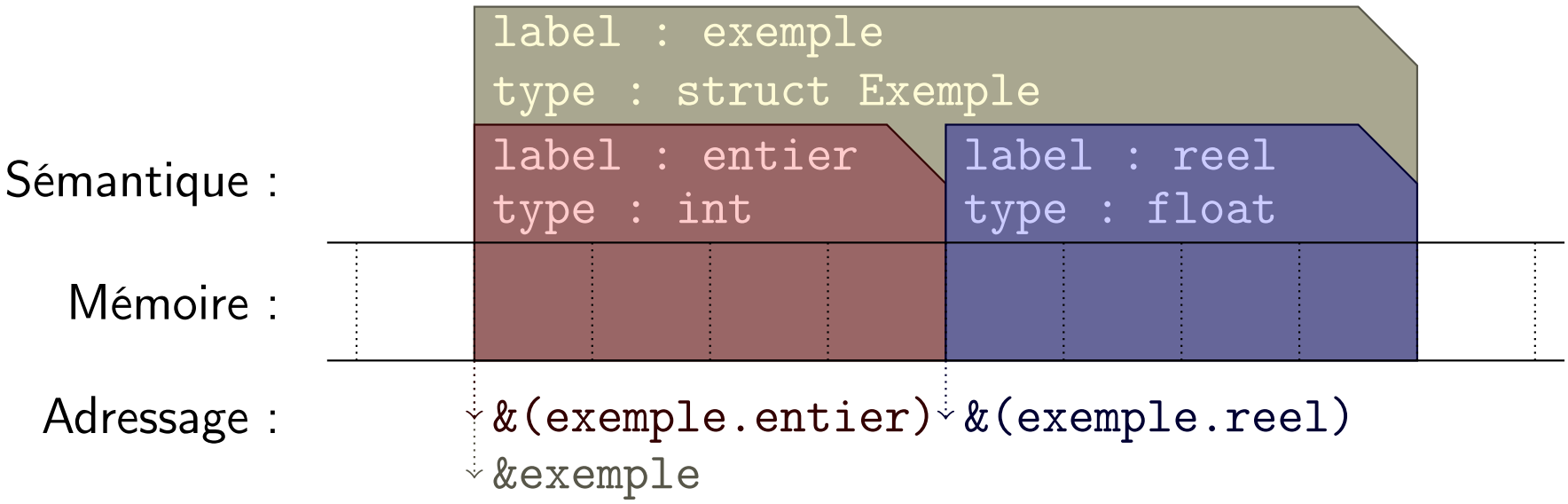
structure : association d'éléments - syntaxe

```c
/* Définition d'une structure 'Nom' : */
struct Nom {
  /* champs : Type nom; */
};

/* Instanciation d'une structure 'Nom' : */
struct Nom variable;
```

structure : association d'éléments - exemple

```
struct Exemple {
  int entier;
  float reel;
};

struct Exemple exemple;
```

structure : association d'éléments - représentation



structure : exemple de liste

```c
/* Schéma de construction d'une Liste */
struct Liste {
  int * elements; /* valeurs de la liste */
  int taille; /* taille de la liste */
};

int main() {
  /* Construction d'une liste */
  struct Liste liste;
  /* accès aux champs de la liste */
  liste.elements;
  liste.taille;

  exit(EXIT_SUCCESS);
}
```

structure : exemple de liste avec synonyme

```c
/* Alias déclaré avant définition */
typedef struct Liste Liste;
/* Schéma de construction d'une Liste */
struct Liste {
  int * elements; /* valeurs de la liste */
  int taille; /* taille de la liste */
};

int main() {
  /* Construction d'une liste */
  Liste liste;
  /* accès aux champs de la liste */
  liste.elements;
  liste.taille;

  exit(EXIT_SUCCESS);
}
```

structure : accès à un champ via pointeur

```
Liste * liste;

/* déréférencement puis accès */
(*liste).elements;

/* version raccourcie */
liste->elements;
```

structure : allocation dynamique

```
Liste * Liste_alloc(int taille) {
  Liste * res = NULL;
  if((res = (Liste *)malloc(sizeof(Liste))) == NULL) {
    fprintf(stderr, "Liste_alloc : Erreur alloc liste\n");
    return NULL;
  }
  res->taille = taille;
  if((res->elements = (int *)calloc(taille, sizeof(int))) ==
  ↪  NULL) {
    free(res);
    fprintf(stderr, "Liste_alloc : Erreur alloc éléments\n");
    return NULL;
  }
  return res;
}
```

structure : allocation dynamique

```c
void Liste_free(Liste ** liste) {
  free((*liste)->elements);
  free(*liste);
  *liste = NULL;
}

int main() {
  /* Construction d'un pointeur de liste */
  Liste * liste = Liste_alloc(4);
  /* accès aux champs de la liste référencée */
  liste->elements;
  liste->taille;

  Liste_free(&liste);

  exit(EXIT_SUCCESS);
}
```

structure et pointeur : opacité de l'implémentation

```c
/* Alias version pointeur */
typedef struct Liste * Liste;
/* Déclaration en amont */
struct Liste;

Liste Liste_alloc(int taille);

void Liste_free(Liste * liste);

int main() {
  /* Construction d'un pointeur de liste */
  Liste liste = Liste_alloc(4);
  /* La liste ne peut plus être manipulée directement */
  /* Il faut maintenant passer par des fonctions */
  Liste_free(&liste);

  exit(EXIT_SUCCESS);
}
```

structure et pointeur : opacité de l'implémentation

```c
/* Votre partie du programme : */

/* pourra être cachée à l'utilisateur de votre code */
/* Schéma de construction d'une Liste */
struct Liste {
  int * elements; /* valeurs de la liste */
  int taille; /* taille de la liste */
};

Liste Liste_alloc(int taille) {
  /* ... */
}

void Liste_free(Liste * liste) {
  /* ... */
}
```

Réduction de la taille mémoire ?

```
struct Personnage {
  unsigned int pointsDeVie;
  unsigned int pointsDeVieMax;
  unsigned int niveau;
  unsigned long experience;
  unsigned char aStatutPoison;
  unsigned char aStatutParalyse;
  unsigned char aStatutEndormi;
  unsigned int attaque;
  unsigned int defense;
  unsigned int attaqueSpe;
  unsigned int defenseSpe;
  unsigned int vitesse;
};
```

Champs de bits : réduction de la taille mémoire

```
struct PersonnageCompress {
  unsigned int pointsDeVie : 10;
  unsigned int pointsDeVieMax : 10;
  unsigned int niveau : 7;
  unsigned long experience : 40;
  unsigned char aStatutPoison : 1;
  unsigned char aStatutParalyse : 1;
  unsigned char aStatutEndormi : 1;
  unsigned int attaque : 10;
  unsigned int defense : 10;
  unsigned int attaqueSpe : 10;
  unsigned int defenseSpe : 10;
  unsigned int vitesse : 10;
};
```

Champs de bits : réduction de la taille mémoire $\star$[1]

```c
int main() {
  printf("%lu\n", sizeof(struct Personnage));
  printf("%lu\n", sizeof(struct PersonnageCompress));
  exit(EXIT_SUCCESS);
}
```

```
48
24
```

Champs de bits : réduction de la taille mémoire $\star^1$

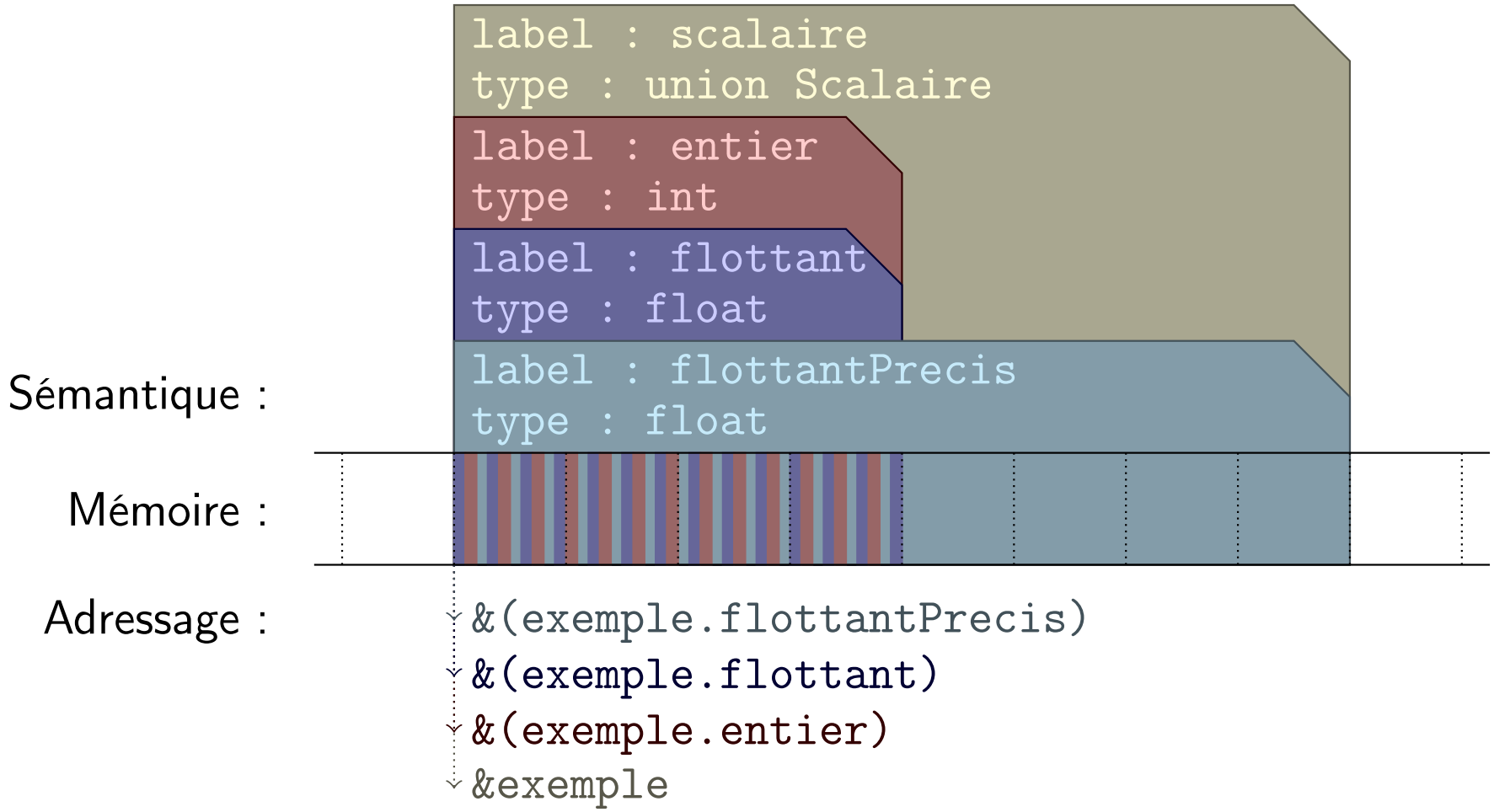
```c
int main() {
  printf("%lu\n", sizeof(struct Personnage));
  printf("%lu\n", sizeof(struct PersonnageCompress));
  exit(EXIT_SUCCESS);
}
```

```
48
24
```

union : regroupement d'éléments - exemple

```
union Scalaire {
  int entier;
  float flottant;
  double flottantPrecis;
};
```

union : regroupement d'éléments - représentation



union : regroupement d'éléments - utilisation $\star^2$

```c
int main() {
  union Scalaire nombre;
  nombre.entier = 42;
  printf("sizeof(Scalaire) : %lu\n", sizeof(union Scalaire));
  printf("entier :           %d\n", nombre.entier);
  printf("flottant :         %g\n", nombre.flottant);
  printf("flottantPrecis :   %g\n", nombre.flottantPrecis);
  nombre.flottant = 42;
  printf("entier :           %d\n", nombre.entier);
  printf("flottant :         %g\n", nombre.flottant);
  printf("flottantPrecis :   %g\n", nombre.flottantPrecis);
  nombre.flottantPrecis = 42;
  printf("entier :           %d\n", nombre.entier);
  printf("flottant :         %g\n", nombre.flottant);
  printf("flottantPrecis :   %g\n", nombre.flottantPrecis);
  exit(EXIT_SUCCESS);
```

union : regroupement d'éléments avec structuration

```
union Scalaire {
  int entier;
  float flottant;
  double flottantPrecis;
  struct {
    float x;
    float y;
  };
};
```

union : regroupement d'éléments avec structuration $\star^3$

```
int main() {
  union Scalaire nombre;
  nombre.x = 42;
  nombre.y = 1337;
  printf("sizeof(Scalaire) : %lu\n", sizeof(union
  ↪  Scalaire));
  printf("entier :          %d\n", nombre.entier);
  printf("flottant :        %g\n", nombre.flottant);
  printf("flottantPrecis :  %g\n",
  ↪  nombre.flottantPrecis);
  printf("(x, y) :          (%g, %g)\n", nombre.x,
  ↪  nombre.y);
  exit(EXIT_SUCCESS);
}
```

enum : type de constantes nommées

```
enum MapItem {
  MAP_VIDE,
  MAP_JOUEUR,
  MAP_ADVERSAIRE,
  MAP_MUR,
  MAP_SORTIE
};

enum MapItem item;
```

enum : type de constantes nommées avec typedef

```
typedef enum {
  MAP_VIDE,
  MAP_JOUEUR,
  MAP_ADVERSAIRE,
  MAP_MUR,
  MAP_SORTIE
} MapItem;

MapItem item;
```

enum : utilisation dans un switch

```
typedef enum {
  ITEM_MOB_DYNAMIC,
  ITEM_MOB_STATIC,
  ITEM_MOB_UNKNOWN
} ItemMobility;

ItemMobility getMapItemMobility(MapItem item) {
  switch(item) {
    case MAP_JOUEUR :
    case MAP_ADVERSAIRE :
    return ITEM_MOB_DYNAMIC;

    case MAP_MUR :
    case MAP_SORTIE :
    return ITEM_MOB_STATIC;

    default :
    return ITEM_MOB_UNKNOWN;
  }
```

enum : assignation de valeurs avec continuité

```
typedef enum {
  MAP_VIDE          = 0,
  MAP_JOUEUR        = 10,
  MAP_ADVERSAIRE /*= 11 */,
  MAP_MUR           = 20,
  MAP_SORTIE     /*= 21 */
} MapItem;
```

enum : assignation de valeurs arbitraires

```
typedef enum {
  MAP_VIDE =        ' ',
  MAP_JOUEUR =      '@',
  MAP_MUR =         '#',
  MAP_ADVERSAIRE =  '£',
  MAP_SORTIE =      'x'
} MapItem;
```

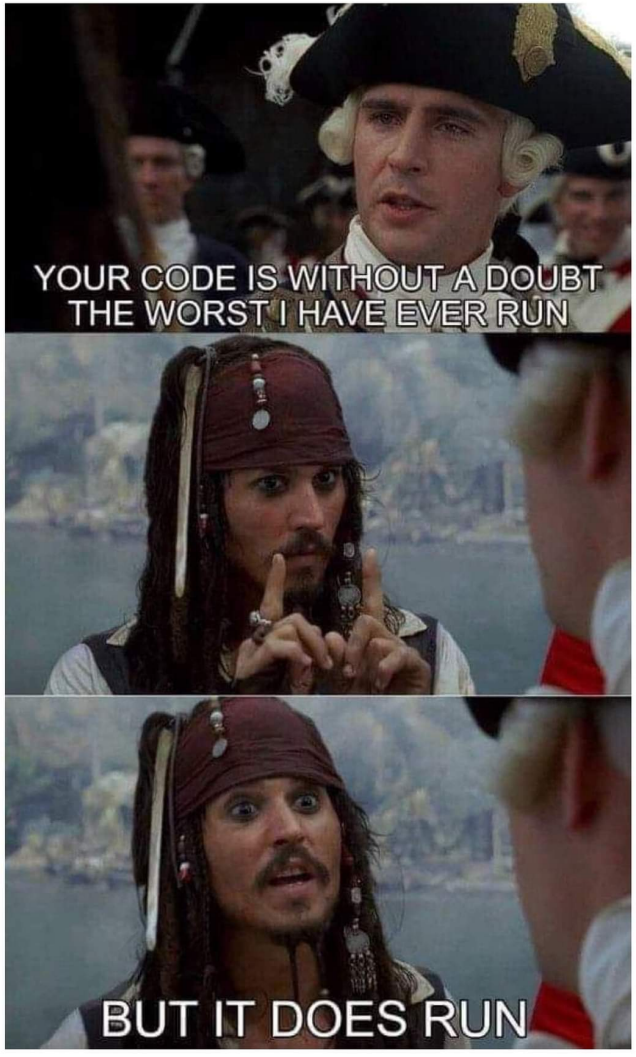
Questions

Avez-vous des questions ?

Exercices

- Travailler sur les exercices sur les structures (section 12) du support de cours.
- Si les exercices de la section 12 sont terminés :
  - Avancer sur les sections 13 et 14.
  - Si cours terminé : Avancer sur le projet.
  - Si projet terminé avec certitude de 21 / 20 : le pousser plus loin.

Annexe


YOUR CODE IS WITHOUT A DOUBT
THE WORST I HAVE EVER RUN
BUT IT DOES RUN