Information, Calcul et Communication

Composante Pratique: Programmation C++

MOOC sem6 : typedef, structure

Questions sur le projet

Comment transformer une *donnée* en une *information* ?

Assembler des données hétérogènes avec `struct`

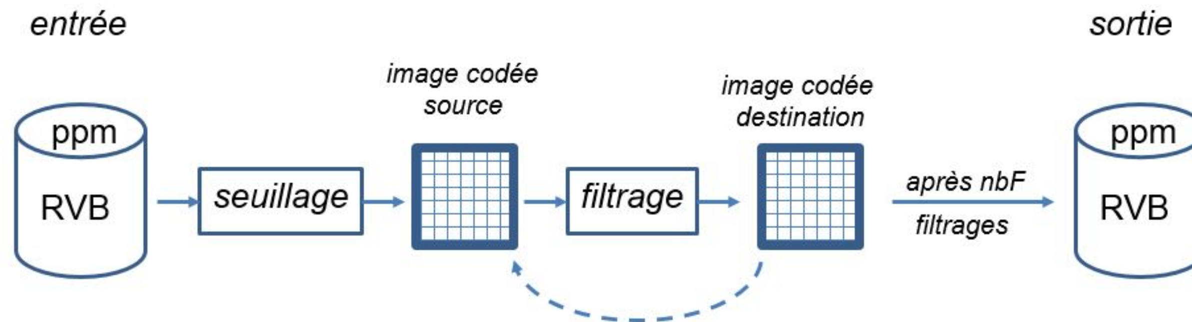
Une structure en mémoire: alignement et padding

Accès fin à la mémoire : bits field

Démo du projet ColoReduce / Questions ?

<u>But</u>

- Pouvoir modifier/reduire les couleurs de n'importe quelle image PPM.
- Mettre au point rigoureusement un problème demandant une décomposition en fonctions (principes d'abstraction et de ré-utilisation).



Comment transformer une donnée en une information ?

Avant de répondre à cette question il faut caractériser ce qu'est une donnée :

Une suite de 0 et de 1 est un **motif binaire**: 0101101100111001
On ne peut rien en faire tant qu'on ne connait pas son **type**: `char, int, double`...

Un **motif binaire** associé à un **type de base** devient une **donnée** manipulable par le processeur avec un ensemble d'opérateurs définis pour ce type.

Ex: 11000001000100000000000000000000 associé au type `float` représente -9.0

Les types de base sont de très bas niveau ; ils sont utiles pour le compilateur pour obtenir un programme exécutable mais ils ne disent rien sur le **but** de ces données.

L'instruction `typedef` permet de transformer une **donnée** en une **information** en définissant un nouveau type synonyme du type de base:

```
typedef float Temperature;
Temperature temp_air(-9.0);
```

Comment transformer une donnée en une information ? (2)

Plus généralement **typedef** permet de créer un nouveau type à partir d'un type que le compilateur connait déjà. C++11 offre une syntaxe plus intutitive avec **using**.

```
typedef type_deja_connu Nouveau_type;   // par convention, un type créé
                                        // par l'utilisateur commence
                                        // par une Majuscule

using Nouveau_type = type_deja_connu;
```

Le nom du nouveau type doit apporter des précisions sur la nature des données, leur signification, leur but etc…

```
typedef array<int,3>     RGB_color;
typedef vector<double>   Vecteur;
typedef vector<Vecteur> Matrice;

RGB_color red ={255,0,0};
Matrice   mat_nulle(3,Vecteur(3));
Vecteur produit_vectoriel(Vecteur v1, Vecteur v2);
```

Assembler des données hétérogènes avec `struct`

Avec **vector**, **array** et un **tableau-à-la-C** on peut seulement rassembler des **données de même type** sous un même identificateur.

Avec le mot clef `struct` on peut rassembler dans un nouveau **type** des **données de types différents.**

C'est l'outil idéal pour structurer les données d'un problème.

```
struct Nom_du_type
{
    type1  identificateur1;
    type2  identificateur2;
    …
};
```

⇒ Bonne pratique: définir D'ABORD le type…

```
Nom_du_type x={val1,val2...};
Nom_du_type y={};
```

… AVANT de créer des variables avec ce type

// init à 0

() [] . ->	* & not ! ++ --	* / %	+ -	< <= > >=	== !=	and &&	or ||	? :

Assembler des données hétérogènes avec `struct` (2)

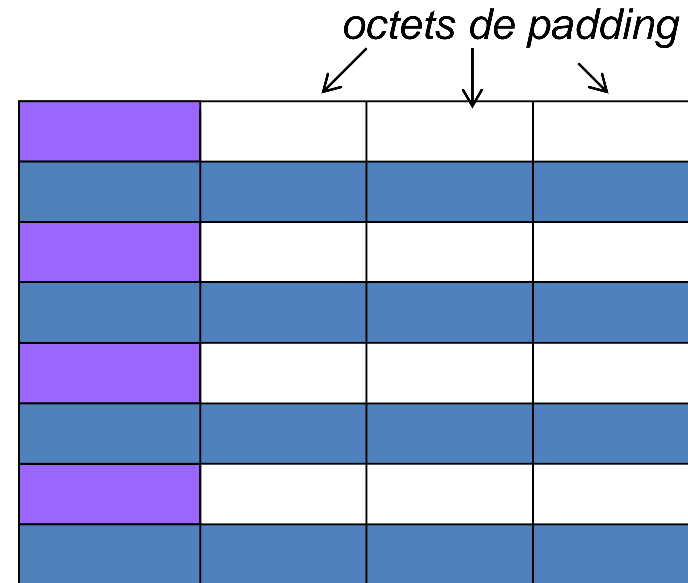
```
struct Personne
{
    string  nom;
    double  taille;
    int     age;
    char    sexe;
};
Personne worker ={"Dupont",1.65,59,'F'};  // C++11: syntaxe aussi utilisable
                                          //        pour l'affectation

++(worker.age);                           // accès à un champ avec .
cout << "Bon anniversaire "               // opérateur de priorité maximum
     << (worker.sexe == 'M')? "M ":"Mme "
     << worker.nom << endl;

vector<Personne> candidat(10);                    // vector de 10 Personne
…
Personne promotion(vector<Personne>& candidat);   // renvoie une Personne

Personne boss=promotion(candidat);  // l'affectation est le seul opérateur autorisé

if(worker == candidat[0])           // interdit ! erreur de compilation
    cout << worker ;                // interdit ! erreur de compilation
```

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

Une structure en mémoire: alignement et padding

l'ordre des champs peut conduire à une occupation mémoire plus importante que la somme de chacun des champs si la taille mémoire de chaque champ ne s'aligne pas sur un mot mémoire

```
struct Fiche1
{
  char    sexe;
  int     age;
  char    permis;
  int     avs;
  char    option;
  float   salaire;
  char    categorie;
  int     nb_heures;
};
```

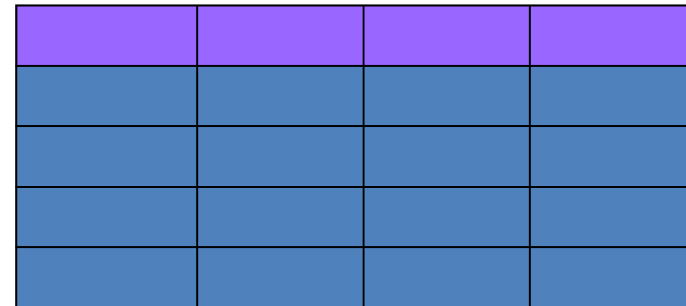


Une structure en mémoire: alignement et padding (2)

l'ordre des champs peut conduire à une occupation mémoire plus importante que la somme de chacun des champs si la taille mémoire de chaque champ ne s'aligne pas sur un mot mémoire

```
struct Fiche2
{
  char   sexe;
  char   permis;
  char   option;
  char   categorie;
  int    age;
  int    avs;
  float  salaire;
  int    nb_heures;
};
```

Alignement **machine 32 bits**



= 5x4 = 20 octets

Une structure en mémoire: alignement et padding (3)

<u>Exemple</u>: Alignements standards sur une **machine 32 bits** : 1 mot = 4 octets

char peut occuper n'importe quel octet

short : sur les octets d'adresse paire

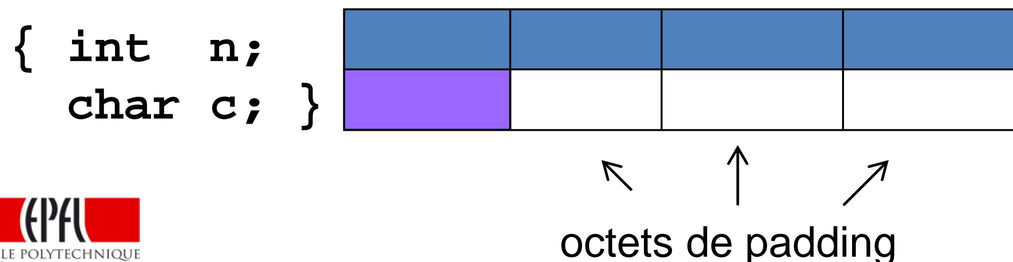
int, long, float: sur les mots

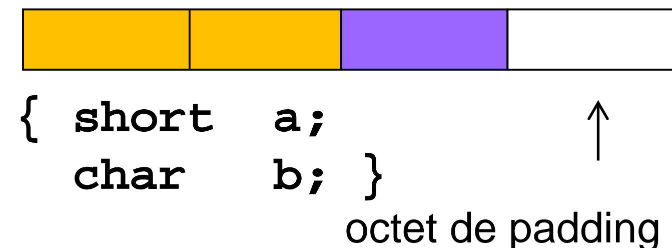
double occupe deux mots

```
{ char  a;
  short b; }
```

Les octets de paddings sont ajoutés aussi en fin de structure pour s'aligner sur le champ de plus grande taille

Pour faciliter la manipulation de tableaux, vector,array



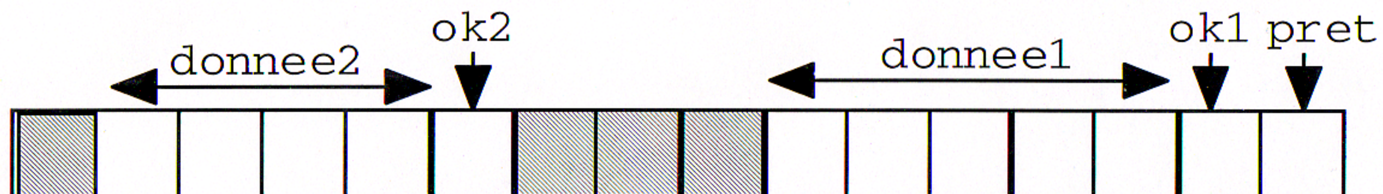


Accès fin à la mémoire : bits field

<u>But</u>: minimiser le temps de communication entre le processeur et les périphériques en transmettant le moins d'octets possible. *On indique le nb de bits par champ*.

Attention*: seulement des types entiers.*
Dépendance machine pour les entier signés

```
struct Etat
{
    unsigned int pret    : 1;   // un seul bit ! C'est un véritable booléen !
    unsigned int ok      : 1;
    int donnee1          : 5;
    int                  : 3;
    unsigned int ok2     : 1;
    int donnee2          : 4;
};
Etat mot;
mot.donnee1 = 13;
```



ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE