PROGRAMMATION ORIENTÉE OBJET

PRESENTATION ET RAPPELS

charles.meunier@ube.fr

PRESENTATION DU MODULE

charles.meunier@ube.fr

PRÉSENTATION

- Quelle est votre expérience en développement ?

- Quelles sont vos attentes par rapport à ce cours ?

OBJECTIFS DU MODULE

- Initiation à la Programmation Orientée Objet (POO)
- Introduction au C++
- Utilisation des outils du développeur :
  - algorithmique,
  - modélisation,
  - IDE,
  - débogage,
  - ...

RÉPARTITION DES ENSEIGNEMENTS

10 CM

pour étudier les concepts de la programmation orientée objet et les caractéristiques du C++

8 TD

pour mettre en pratique les notions abordées en cours

4 TP

pour réaliser un projet en C++

Cours et sujets disponibles sur https://lamarmotte.Info

charles.meunier@ube.fr

EVALUATION

CONTRÔLE CONTINU

- Evaluations en début de TD sur les CM et TD précédents
- TD notés
- Projet en TP

CONSEILS

- Prenez des notes
- Posez des questions
- Réalisez des mini-projets à la maison

PETIT RETOUR EN ARRIERE

charles.meunier@ube.fr

VARIABLES

- **Qu'est-ce qu'une variable ?**
  Un espace en mémoire dans lequel on stocke une donnée

- **Quels sont les types de variables que vous connaissez en C ?**
  [unsigned] char, [unsigned][long] int, float, [long] double, pointeurs

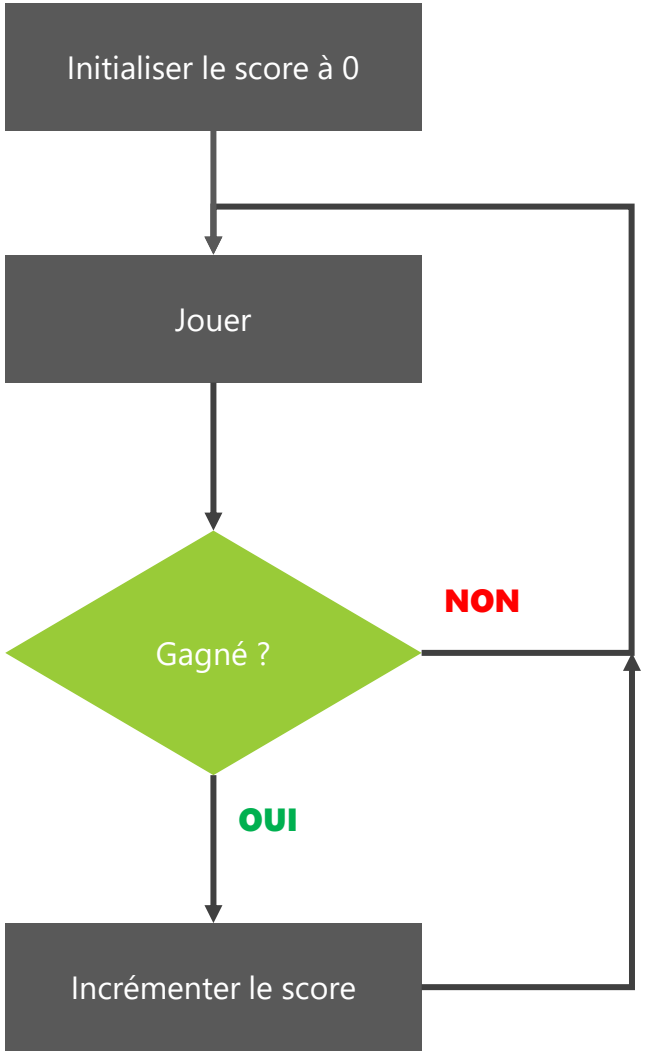
- **Comment déclare-t-on une variable de type entier ?**
  ```
  int variableDeTypeEntier = 0;
  ```

- **Comment nomme-t-on une variable ?**
  En fonction de ce qu'elle contient. Par exemple, une variable entière qui contiendra le score de l'utilisateur se déclarera :
  ```
  int score = 0;
  ```

  Prenez l'habitude de travailler en anglais.

VARIABLES GLOBALES

BOUCLES ET TESTS




```
unsigned int score = 0;

while(1)
{
    unsigned char won = play();

    if(won == 1)
        score++;
}
```

BOUCLES ET TESTS

Calculez la consommation électrique moyenne d'un foyer à partir des consommations quotidiennes mesurées sur les 30 derniers jours.

Soit consoQuotidiennes un tableau de 30 entiers
Soit consoTotale un entier initialisé à 0

Pour chaque élément d'indice i du tableau
Faire
    consoTotale += consoQuotidiennes [i];
Fin pour

Retourner consoTotale / 30;

```
unsigned int dailyConsumptions[30] = { 1753, 2011, ... };
unsigned int totalConsumption = 0;
unsigned int iConsumption;

for(iConsumption = 0; iConsumption < 30; ++iConsumtion)
{
    totalConsumption += dailyConsumption[iConsumption];
}

return totalConsumption / 30;
```

ATTENTION À LA DIVISION ENTIÈRE

FONCTIONS

- **A quoi servent les fonctions de manière générale ?**
  - A découper le code en sections élémentaires
  - A factoriser le code

- **Comment nomme-t-on une fonction ?**
  En fonction de ce qu'elle fait. Par exemple, une fonction qui envoie une email :

```
int sendMail(char* address, char* subjet, char* body)
{
    ...
}
```

- **Comment appelle-t-on une fonction ?**

```
sendMail("70pi1@etu.u-bourgogne.fr", "POO - Epreuve", "Le sujet portera sur...");
```

FONCTIONS

- **A quoi sert la fonction main ?**
  C'est le point d'entrée du programme, la toute première fonction qui sera appelée automatiquement au démarrage du programme.

```
int main(int argc, char** argv)
{
}
```

- **A quoi correspondent les paramètres argc et argv de la fonction main ?**
  - **argc** correspond au nombre d'arguments (arguments count) transmis à l'exécutable du programme
  - **argv** contient la liste de ses arguments (arguments values). C'est un `char**`, c'est à dire un tableau de chaines de caractères.

- **Pourquoi la fonction main retourne un int ?**
  Cette valeur de retour permet d'indiquer au système d'exploitation si le programme s'est terminé correctement (0) ou s'il y a eu un problème (valeur différente de 0)

STRUCTURE

- **A quoi servent les structures ?**
  A définir un nouveau type qui regroupera des données de types, éventuellement, différents au sein d'une structure de donnée cohérente.

- **Comment déclare-t-on une structure ?**

```
struct Point {
    int x;
    int y;
    int z;
}
```

- **Comment instancie-t-on une structure ?**

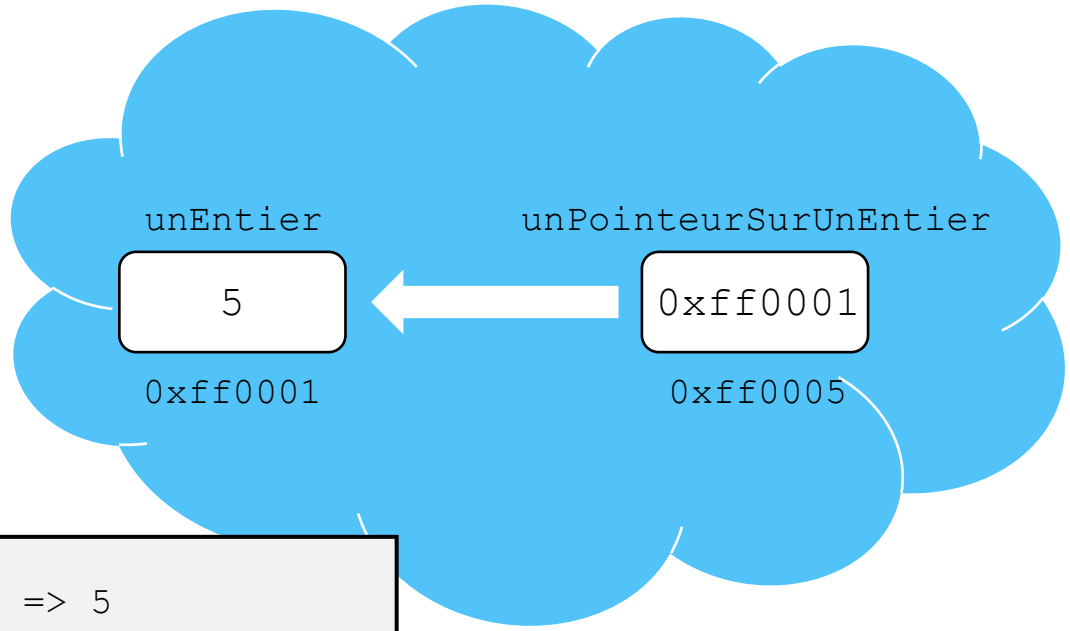
```
// Sans initialisation (attention, les membres x, y et z de la structure
// auront une valeur aléatoire)
struct Point pt;

// Avec initialisation (recommandée)
struct Point origin = {0, 0, 0};
```

POINTEURS

- **Qu'est-ce qu'un pointeur ?**
  Une variable qui contient une adresse en mémoire.

  Exemple :

```
int unEntier = 5;
int* unPointeurSurUnEntier = &unEntier;
*unPointeurSurUnEntier = 18;
```




```
unEntier                => 5

&unEntier               => 0xFF0001

unPointeurSurUnEntier   => 0xFF0001

&unPointeurSurUnEntier  => 0xFF0005

*unPointeurSurUnEntier  => 5
```

POINTEURS

- **Dans quels cas utilise-t-on des pointeurs ?**
  - Les tableaux
  - Les chaines de caractères
  - Les passages en paramètres de données complexes
  - L'allocation dynamique
  - ...

TABLEAUX

- **Qu'est-ce qu'un tableau ?**
  Les tableaux n'existent pas nativement en C.

  C'est une suite de données, toutes du même type, contiguës en mémoire et dont on connait l'adresse du premier élément.

  Exemple :

```
int tab[4] = { 10, 20, 30, 40 };
```

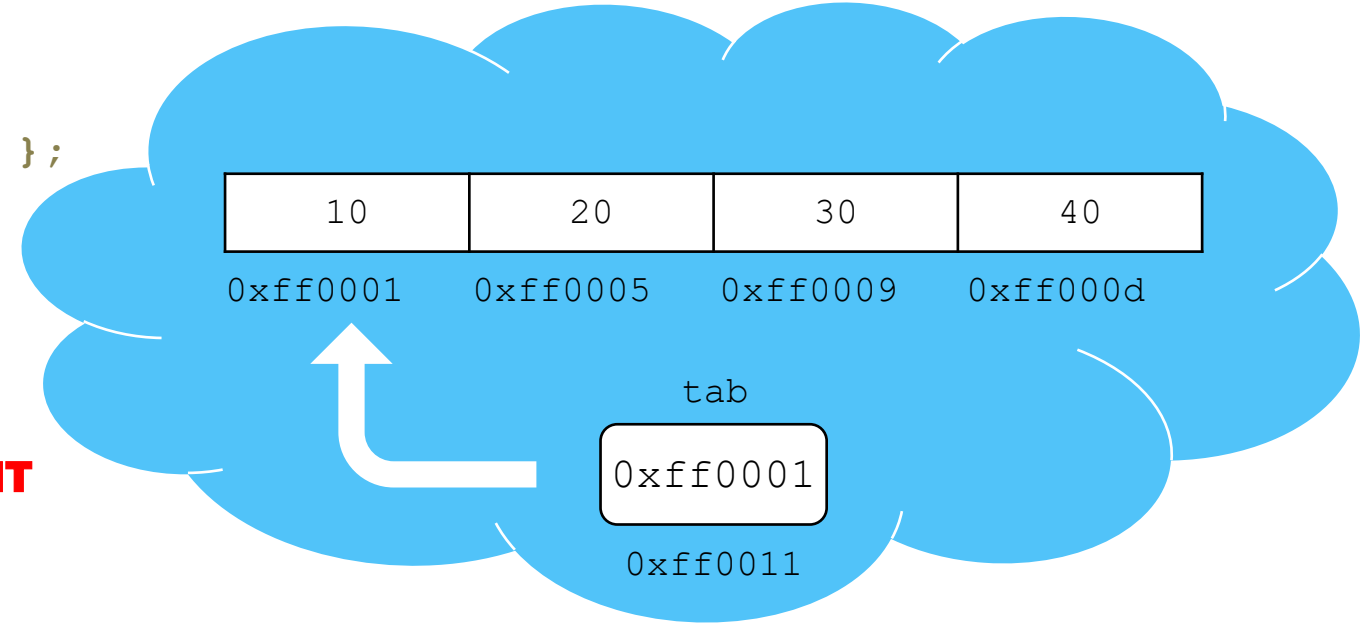


TABLEAU
=
POINTEUR SUR LE PREMIER ELEMENT

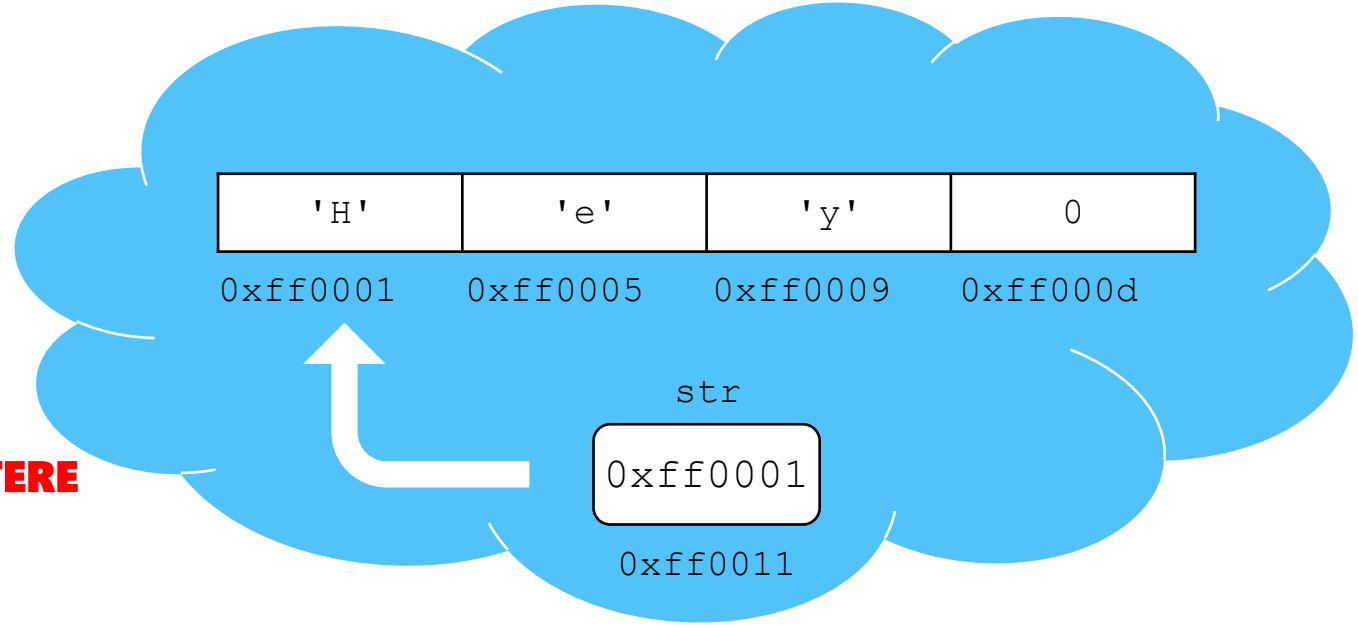
CHAINE DE CARACTERES

- **Qu'est-ce qu'une chaine de caractères ?**
  Les chaines de caractères n'existent pas nativement en C.

  C'est une suite de caractères contiguës en mémoire qui se termine par un 0 (ASCII-Z) et dont on connait l'adresse du premier caractère.

  Exemple :

```
char* str = "Hey";
```



CHAINE DE CARACTERES
=
POINTEUR SUR LE PREMIER CARACTERE

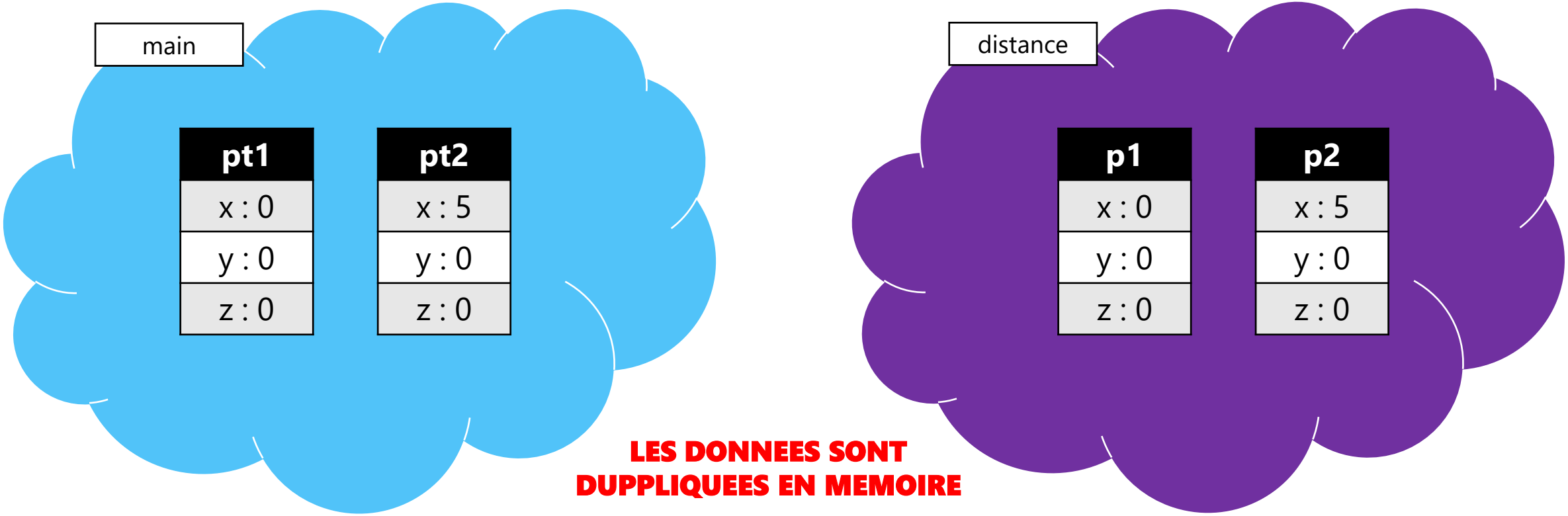
PASSAGE EN PARAMETRE DE DONNEE COMPLEXE

- **Passage par valeur (par recopie)**

```
double distance(struct Point p1, struct Point p2)
{
    return sqrt(pow(p2.x - p1.x, 2) + pow(p2.y - p1.y, 2));
}

int main(int argc, char** argv)
{
    struct Point pt1 = { 0, 0, 0 };
    struct Point pt2 = { 5, 0, 0 };

    double d = distance(pt1, pt2);

    return 0;
}
```

PASSAGE EN PARAMETRE DE DONNEE COMPLEXE

- **Passage par valeur (par recopie)**



LES DONNEES SONT DUPPLIQUEES EN MEMOIRE

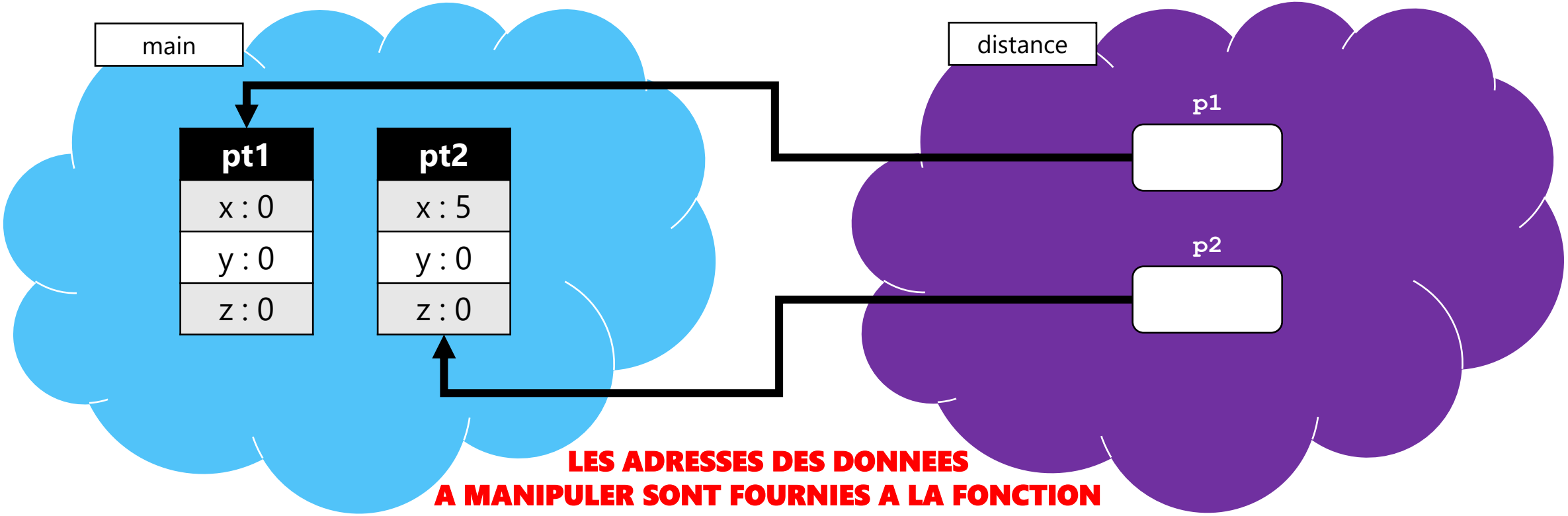
PASSAGE EN PARAMETRE DE DONNEE COMPLEXE

- **Passage par adresse**

```
double distance(struct Point* p1, struct Point* p2)
{
    return sqrt(pow(p2->x - p1->x, 2) + pow(p2->y - p1->y, 2));
}

int main(int argc, char** argv)
{
    struct Point pt1 = { 0, 0, 0 };
    struct Point pt2 = { 5, 0, 0 };

    double d = distance(&pt1, &pt2);

    return 0;
}
```

PASSAGE EN PARAMETRE DE DONNEE COMPLEXE

- **Passage par valeur (par recopie)**



.H ET .C

Le .h contient les déclarations (fonctions, constantes, ...)

bool.h

```h
#define TRUE 1
#define FALSE 0
#define BOOL unsigned char

BOOL bool_and(BOOL b1, BOOL b2);

BOOL bool_or(BOOL b1, BOOL b2);
```

Le .c contient les définitions des fonctions (le code)

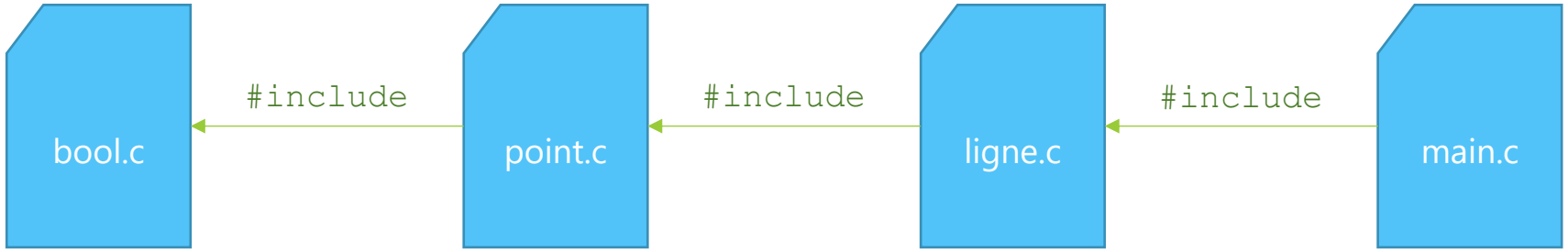
bool.c

```c
#include "bool.h"

BOOL bool_and(BOOL b1, BOOL b2)
{
    return b1 && b2;
}

BOOL bool_or(BOOL b1, BOOL b2)
{
    return b1 || b2;
}
```

.H ET .C : POURQUOI ?

- Gagner en lisibilité : les prototypes de fonctions ne sont pas noyés dans le code

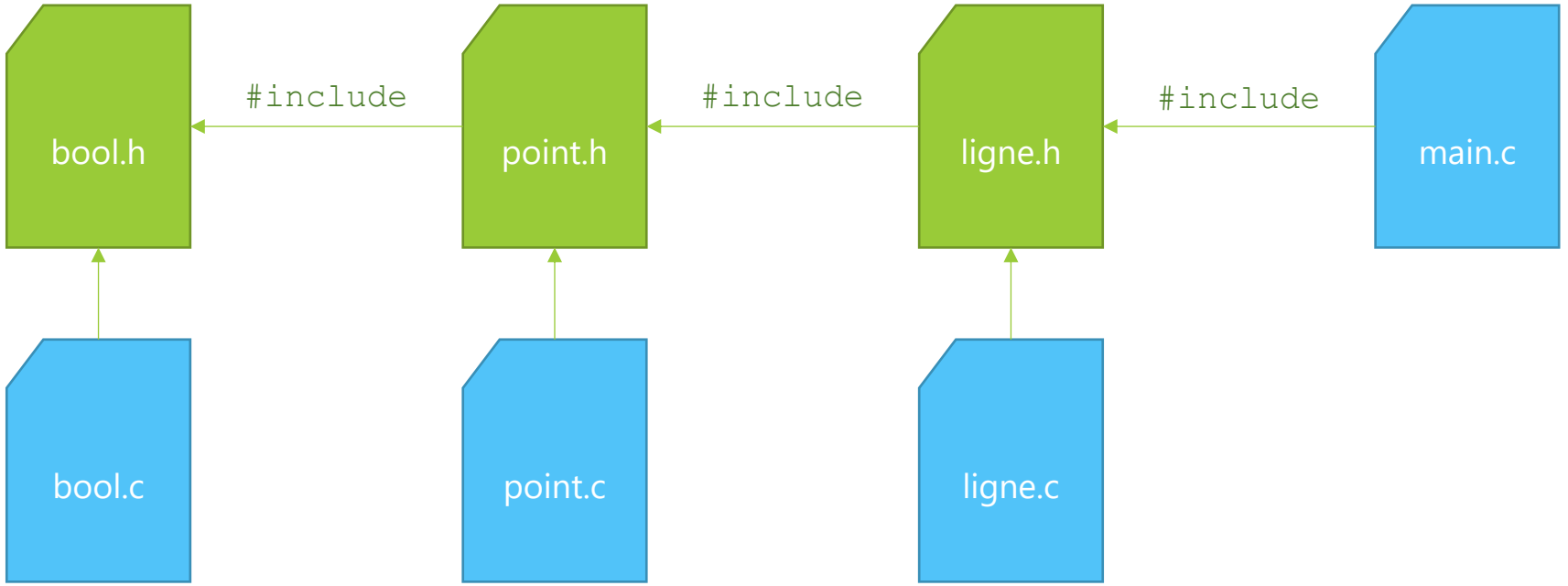
- Optimiser les temps de compilation



MODIFIER BOOL.C ENTRAÎNE LA RECOMPILATION DE TOUS LES FICHIERS QUI EN DÉPENDENT

.H ET .C : POURQUOI ?

- Optimiser les temps de compilation



MODIFIER BOOL.C N'ENTRAÎNE QUE LA RECOMPILATION DE BOOL.C

INSTRUCTION DU PRÉPROCESSEUR

- **#include**
  Recopie le contenu du fichier indiqué dans le fichier courant

- **#define**
  Définit une étiquette qui sera remplacée dans le code par la valeur associée

```
#define TRUE 1
...
if(a == TRUE)
{
    ...
}
```

APRÈS PRÉPROCESSEUR

```
if(a == 1)
{
    ...
}
```

- **#ifdef / #ifndef / #endif**
  Définit une portion de code conditionnelle qui sera ou non présente dans le code compilé en fonction de la validité de la condition

POUR TERMINER CE PREMIER COURS

charles.meunier@ube.fr

UN EXERCICE

Faites tourner ce programme à la main (sur feuille sans ordinateur) et déterminez ce qu'affichera la console à la fin de l'exécution.

```
char valeur[] = "Frite";
char* p1 = valeur;
char** p2 = &p1;

p1[0] -= 3;
*(p1 + 1) = 'h';
valeur[3] = *(p1 + 2) + 7;
p1[4] = *(valeur + 4) + (*(p2[0] + 2) - 'a') + 6;

printf("%s", valeur);
```