Langages de programmation

Cours 7

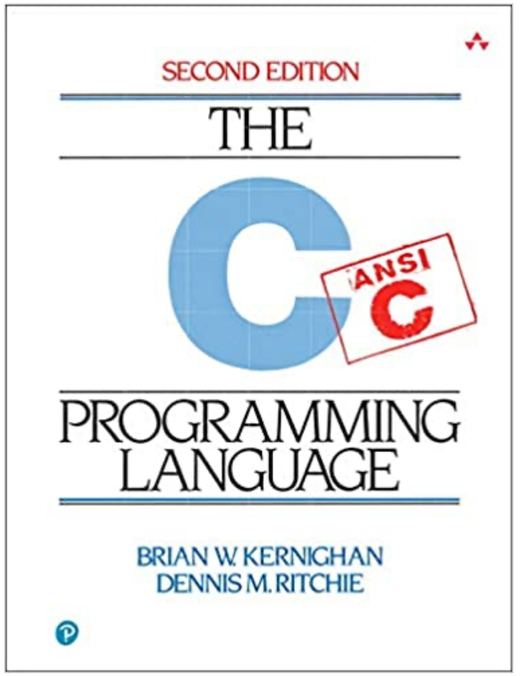
Jean-Jacques Lévy

jean-jacques.levy@inria.fr

http://jeanjacqueslevy.net/lp-prog

Plan

- enregistrements en C (`struct, union`)
- classes en C++
- programmation procédurale en C++
- classes dérivées
- programmation objet en C++
- héritage
- exemples



A Lire: tutorial sur le langage C++ `http://www.programiz.com/c-programming`

Enregistrements (1/4)

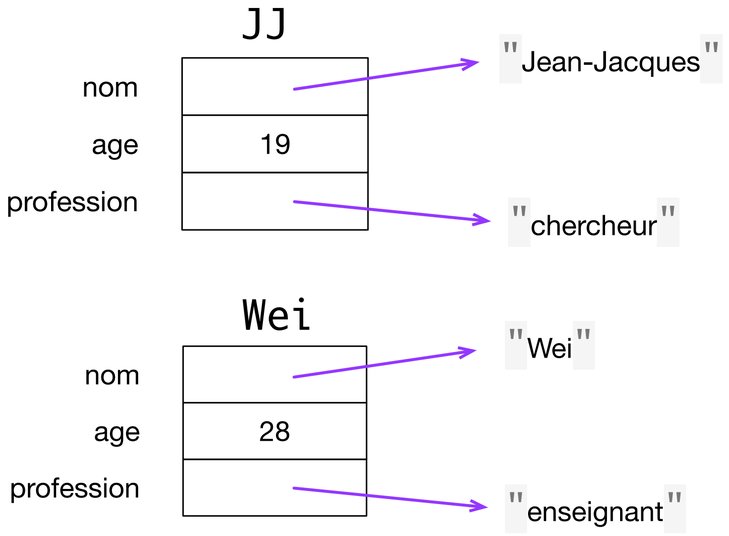
- les enregistrements (*records*) en C se font avec la commande `struct`

```c
struct personne {
    char *nom;
    int age;
    char *profession;
};
```

```c
void printPersonne (struct personne *p) {
    printf ("%s, %d ans, %s\n", p->nom, p->age, p->profession);
}
```

```c
struct personne JJ, Wei;

int main () {
    JJ.nom = "Jean-Jacques"; JJ.age = 19;
    JJ.profession = "chercheur";
    Wei.nom = "Wei"; Wei.age = 28;
    Wei.profession = "enseignant";
    printPersonne (&JJ);
    printPersonne (&Wei);
    return 0;
}
```



- en C, les 2 notations suivantes sont équivalentes

`p->nom` ≡ `(*p).nom`

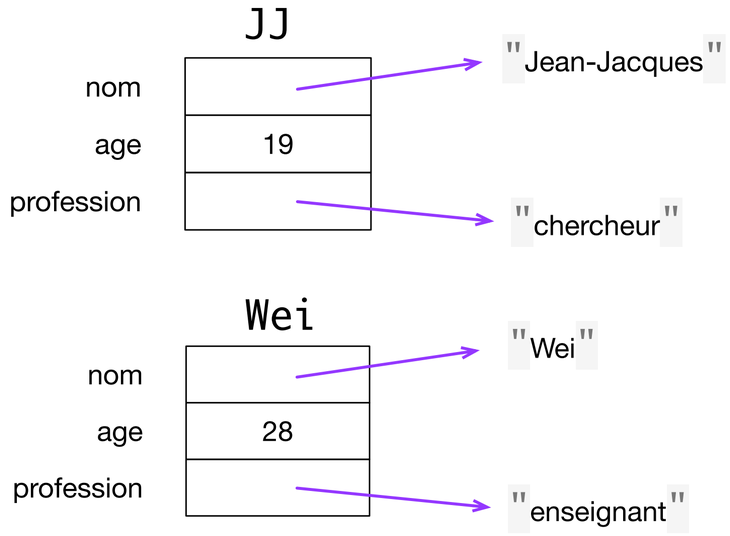
Enregistrements (2/4)

- on peut aussi déclarer simultanément le type et des enregistrements

```
struct personne {
    char *nom;
    int age;
    char *profession;
} JJ, Wei;
```

```
void printPersonne (struct personne *p) {
    printf ("%s, %d ans, %s\n", p->nom, p->age, p->profession);
}
```

```
int main () {
    JJ.nom = "Jean-Jacques"; JJ.age = 19;
    JJ.profession = "chercheur";
    Wei.nom = "Wei"; Wei.age = 28;
    Wei.profession = "enseignant";
    printPersonne (&JJ);
    printPersonne (&Wei);
    return 0;
}
```



- en C, les 2 notations suivantes sont équivalentes

`p->nom` ≡ `(*p).nom`

Enregistrements (3/4)

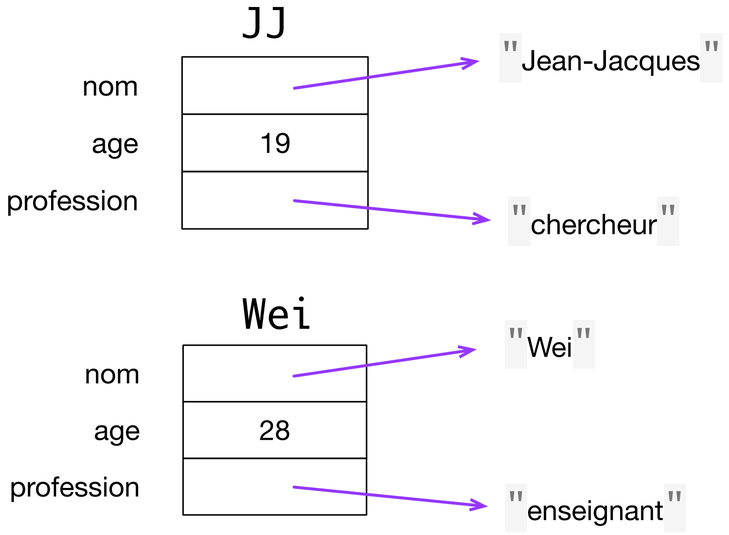
- on peut carrément définir un nouvel identifieur de type

```
typedef struct personne {
    char *nom;
    int age;
    char *profession;
} Personne;


void printPersonne (Personne *p) {
    printf ("%s, %d ans, %s\n", p->nom, p->age, p->profession);
}


Personne JJ, Wei;

int main () {
    JJ.nom = "Jean-Jacques"; JJ.age = 19;
    JJ.profession = "chercheur";
    Wei.nom = "Wei"; Wei.age = 28;
    Wei.profession = "enseignant";
    printPersonne (&JJ);
    printPersonne (&Wei);
    return 0;
}
```



- en C, les 2 notations suivantes sont équivalentes

`p->nom` ≡ `(*p).nom`

Enregistrements (4/4)

- et définir un constructeur de tels enregistrements

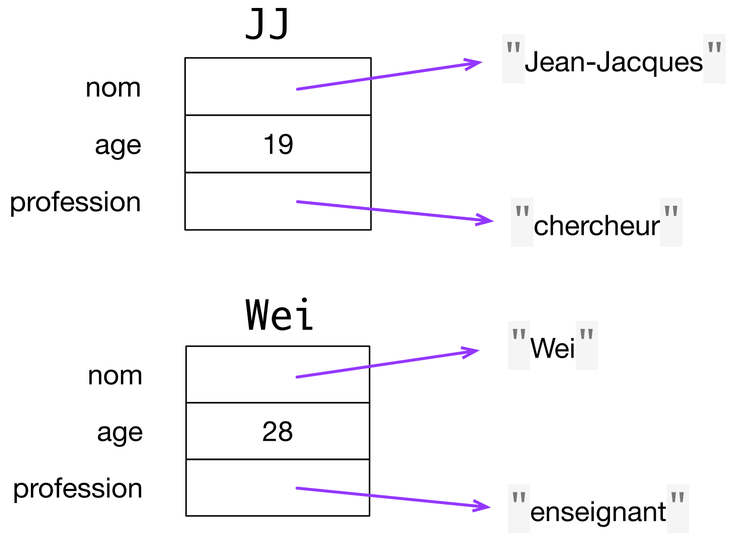
```c
#include <stdio.h>
#include <stdlib.h>

typedef struct personne {
    char *nom;
    int age;
    char *profession;
} Personne;

Personne *unePersonne (char s[], int age, char job[]) {
    Personne *p = malloc(sizeof(struct personne));;
    p->nom = s; p->age = age; p->profession = job;
    return p;
}

void printPersonne (Personne *p) {
    printf ("%s, %d ans, %s\n", p->nom, p->age, p->profession);
}

int main () {
    Personne *JJ = unePersonne ("Jean-Jacques", 19, "chercheur");
    Personne *Wei = unePersonne ("Wei", 28, "enseignant");
    printPersonne (JJ);
    printPersonne (Wei);
    return 0;
}
```

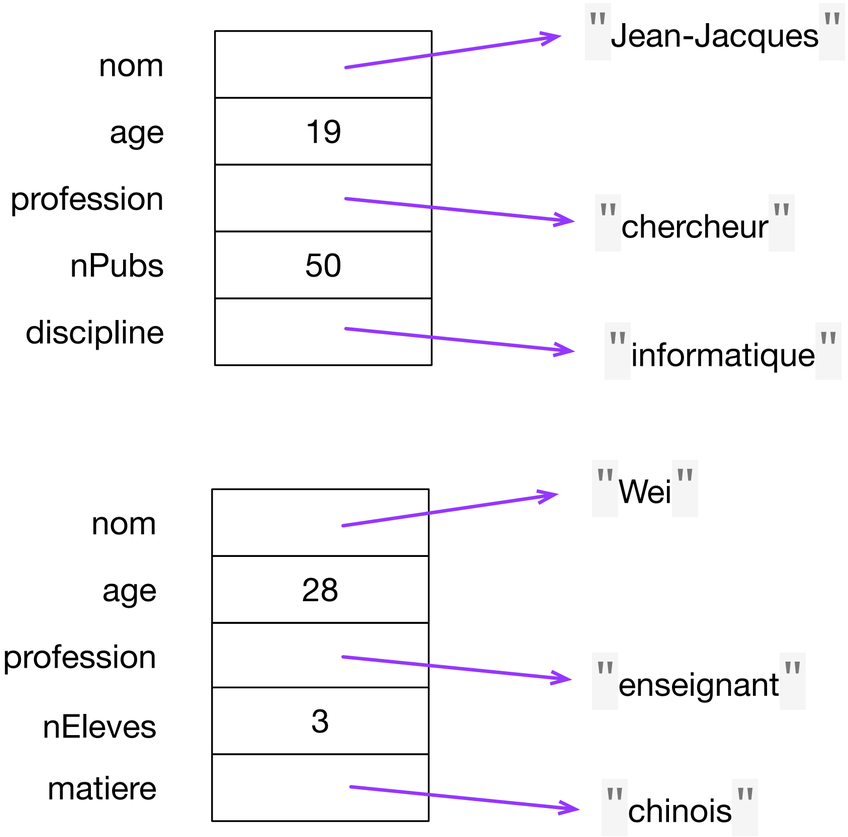


Enregistrements avec variants (1/3)

- on peut définir un variant avec la commande `union`

```c
#include <stdio.h>
#include <stdlib.h>
#include <strings.h>

typedef struct personne {
    char *nom;
    int age;
    char *profession;
    union {
        struct {int nPubs; char *discipline; };
        struct {int nEleves; char *matiere; };
    };
} Personne;
```



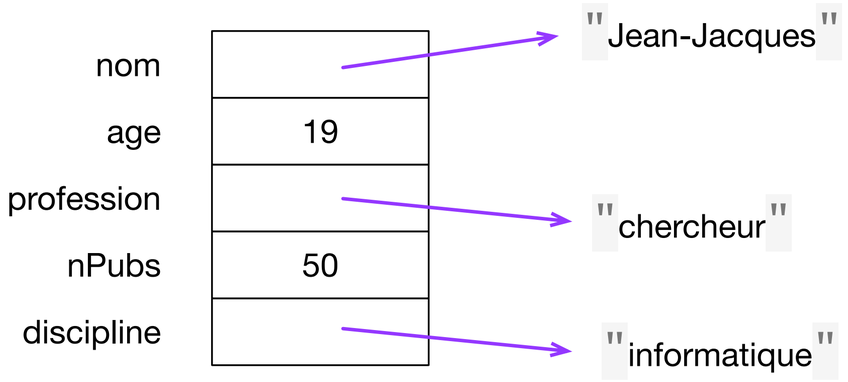
Enregistrements avec variants (2/3)

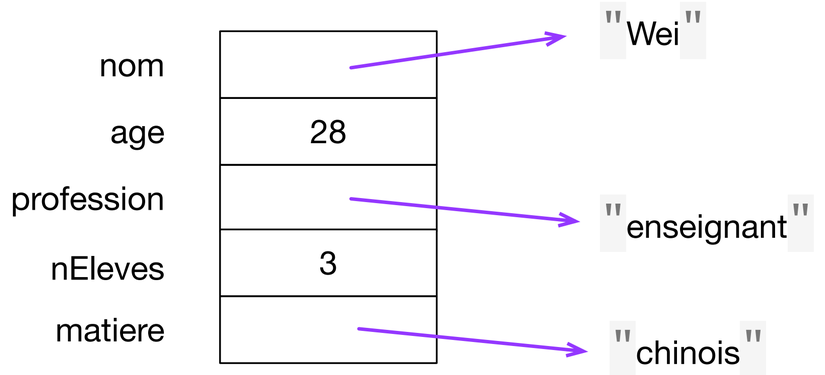
- et définir un constructeur de tels enregistrements

```
Personne *unEnseignant (char n[], int age, int nbe, char s[]) {
    Personne *p = malloc(sizeof(struct personne));;
    p->nom = n; p->age = age; p->profession = "enseignant";
    p->nEleves = nbe; p->matiere = s;
    return p;
}

Personne *unChercheur (char n[], int age, int nbpubs, char s[]) {
    Personne *p = malloc(sizeof(struct personne));;
    p->nom = n; p->age = age; p->profession = "chercheur";
    p->nPubs = nbpubs; p->discipline = s;
    return p;
}

void printPersonne (Personne *p) {
    if (strcmp (p->profession, "chercheur") == 0)
       printf ("%s, %d ans, chercheur en %s, %d publications\n",
               p->nom, p->age, p->discipline, p->nPubs);
    else
       printf ("%s, %d ans, enseignant de %s, %d eleves\n",
               p->nom, p->age, p->matiere, p->nEleves);
}
```

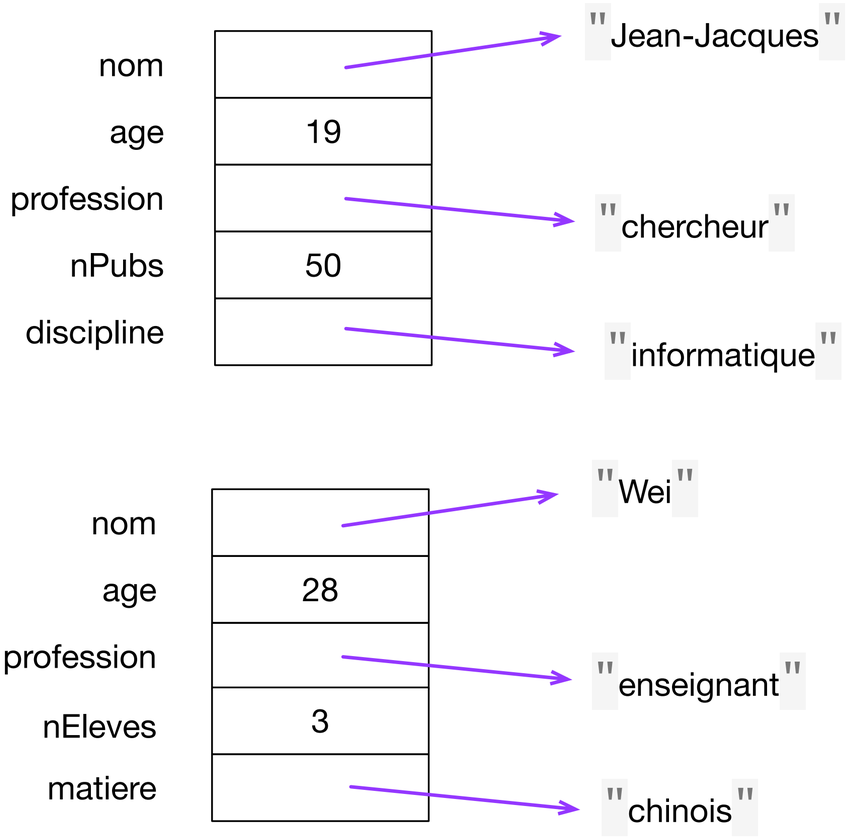




Enregistrements avec variants (3/3)

- et définir un constructeur de tels enregistrements

```
int main () {
    Personne *JJ = unChercheur ("Jean-Jacques", 19, 50, "informatique");
    Personne *Wei = unEnseignant ("Wei", 28, 3, "chinois");
    printPersonne (JJ);
    printPersonne (Wei);
    return 0;
}
```



C++

C++

Stroustrup

=

C

programmation système

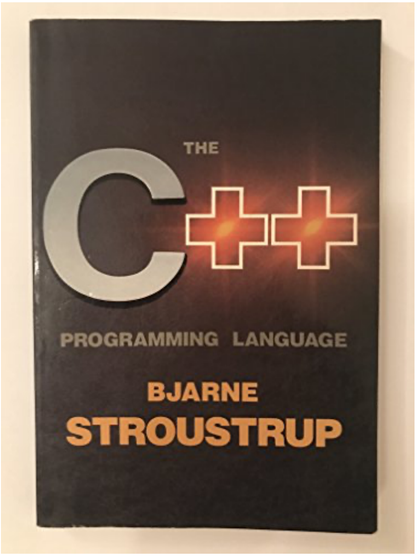
Ritchie

+

Simula

programmation objet

Nygaard



- Smalltalk aussi ancêtre de la programmation objet

Débuts en C++

[rappel]

- le suffixe des noms de fichiers est `.C` au lieu de `.c`
- C++ a les mêmes types de base et opérateurs que C
- les entrées-sorties sont différentes

flux de sortie

```
#include <iostream>
using namespace std;

int main() {
    cout << "Bonjour tout le monde" << endl;
    return 0;
}
```

flux d'entrée

```
int main() {
    char a;
    int num;
    cout << "Entrez caractère et nombre: " ;
    cin >> a >> num;
    cout << "a = " << a << "; ";
    cout << "num = " << num << endl;
    return 0;
}
```

end line

[on n'a pas à spécifier le type du format comme dans `printf` (merci à la programmation objet !) — voir plus tard]

Débuts en C++

[rappel]

- l'allocation mémoire se fait avec la primitive `new` (plus besoin de `malloc` et `sizeof` !)

```
int main() {
    int *a = new int[10];
    for (int i = 0; i < 10; ++i) a[i] = i*i;

    cout << a << " " << endl;
    for (int i = 0; i < 10; ++i)
       cout << a[i] << " ";
    return 0;
}
```

ne pas oublier d'initialiser le tableau

- la dé-allocation mémoire se fait avec la primitive `free` (qui existe aussi en C)

[très peu utilisé à cause des alias — problème général du ramasse-miettes (*garbage collector*)]

Surcharge des fonctions

[rappel]

- les fonctions peuvent être surchargées (*overloading*)
- elles peuvent avoir un code différent selon le type ou le nombre de leurs arguments

```
int absolu (int x) {
    if (x < 0)
      return -x;
    else
      return x;
}

double absolu (double x) {
    if (x < 0)
      return -x;
    else
      return x;
}

int main() {
    cout << absolu(22) << " " << absolu (-42) << endl;
    cout << absolu(3.14) << " " << absolu (-1.28) << endl;
}
```

résultat entier

résultat réel flottant 64bits

C++ en tant qu'extension de C

- en C++, on peut écrire un programme C avec des `struct et union`

```
#include <iostream>
using namespace std;

typedef struct personne {
    const char *nom;
    int age;
    const char *profession;
    union {
        struct {int nPubs; const char *discipline; };
        struct {int nEleves; const char *matiere; };
    };
} Personne;

Personne *unEnseignant (const char n[], int age, int nbe,  const char s[]) {
    Personne *p = new (Personne);;
    p->nom = n; p->age = age; p->profession = "enseignant";
    p->nEleves = nbe; p->matiere = s;
    return p;
}

Personne *unChercheur (const char n[], int age, int nbpubs, const char s[]) {
    Personne *p = new (Personne);;
    p->nom = n; p->age = age; p->profession = "chercheur";
    p->nPubs = nbpubs; p->discipline = s;
    return p;
}
```

C++ en tant qu'extension de C

- en C++, on peut écrire un programme C avec des `struct`

```
void printPersonne (Personne *p) {
    if (strcmp (p->profession, "chercheur") == 0)
       cout << p->nom << ", " << p->age << " ans, chercheur en "
            << p->discipline << ", " << p->nPubs << endl;
    else
       cout << p->nom << ", " << p->age << " ans, enseignant de "
            << p->matiere << ", " << p->nEleves << endl;
}

int main () {
    Personne *JJ = unChercheur ("Jean-Jacques", 19, 50, "informatique");
    Personne *Wei = unEnseignant ("Wei", 28, 3, "chinois");
    printPersonne (JJ);
    printPersonne (Wei);
    return 0;
}
```

- et chaque fois qu'on introduira une nouvelle profession, il faudra changer à **plusieurs** endroits du programme:
  la structure Personne, un nouveau constructeur, la fonction d'impression, etc.

Classes et Objets

- le style de programmation procédurale de C n'est pas le style de C++

- en C++, on utilise des classes et des objets

```
class Personne {
   public:
    string nom;
    int age;

    Personne (string n, int a) { nom = n; age = a; };
};
```

```
int main () {
    Personne JJ = Personne ("Jean-Jacques", 19);
    Personne Wei = Personne ("Wei", 28);

    printPersonne (&JJ);
    printPersonne (&Wei);
    return 0;
}
```

JJ et Wei sont des **objets** de la classe Personne

```
void printPersonne (Personne *p) {
    cout << p->nom << ", " << p->age << " ans\n";
}
```

Classes et Objets

- une classe contient des données (*data*) et des fonctions publiques ou privées (par défaut).

- ces fonctions sont souvent appelés les méthodes de l'objet

```
class Personne {
   public:
    string nom;
    int age;

    Personne (string n, int a) { nom = n; age = a; };

    string str () {
      return nom + string(", ") + to_string(age) + string(" ans");
    }
};
```

données (publiques)

constructeur

méthode pour transformer en `string`

```
int main () {
    Personne JJ = Personne ("Jean-Jacques", 19);
    Personne Wei = Personne ("Wei", 28);
    cout << JJ.str() << endl;
    cout << Wei.str() << endl;
    return 0;
}
```

Classes et Objets

- une sous-classe spécifie les données et méthodes de la classe dont elle dérive

```
class Enseignant : Personne {
   public:
    int nbEleves;
    string matiere;

    Enseignant (string n, int a, string s, int ne) :
        Personne(n, a) { matiere = s; nbEleves = ne; }

    string str () {
      return Personne::str()
          + string(", enseignant de ") + matiere + string(", ")
          + to_string(nbEleves) + string(" élèves");
    }
};
```

nouvelles données

constructeur

redéfinition de la méthode `str`

```
int main () {
    Enseignant Wei = Enseignant ("Wei", 28, "chinois", 3);
    cout << Wei.str() << '\n';
    return 0;
}
```

Classes et Objets

- une classe contient des données (*data*) et des fonctions publiques ou privées (par défaut).

- ces fonctions sont souvent appelés les méthodes de l'objet

```
class Chercheur : Personne {
   public:
    int nbPubs;
    string discipline;

    Chercheur (string n, int a, string s, int ne) :
      Personne(n, a) { discipline = s; nbPubs = ne; }

    string str () {
      return Personne::str()
          + string(", chercheur en ") + discipline + string(", ")
          + to_string(nbPubs) + string(" publications");
    }
};
```

nouvelles données

constructeur

redéfinition de la méthode `str`

```
int main () {
    Chercheur JJ = Chercheur ("Jean-Jacques", 19, "informatique", 50);
    cout << JJ.str() << '\n';
    return 0;
}
```

Programmation objet

- chaque fois qu'on introduit une nouvelle profession, il faut changer à **un seul** endroit du programme: en rajoutant une nouvelle sous-classe.

- la programmation objet a un comportement **incrémental** par rapport aux données

- si on veut changer ou rajouter une fonction, il faudra le faire à **plusieurs** endroits du programme

	ajout de données	ajout de fonctions
programmation objet	+	-
programmation procédurale	-	+

programmation objet == programmation par les données

Exercices

Exercice Dans l'exemple précédent, que donne la ligne suivante ?

```
Chercheur JJ = Personne ("Jean-Jacques", 19, "informatique", 50);
```

Exercice Dans l'exemple précédent, que donnent les lignes suivantes ?

```
Personne JJ = Chercheur ("Jean-Jacques", 19, "informatique", 50);
cout << JJ.str() << '\n';
```

Exercice Dans la définition des différentes classes précédentes, que se passe-t-il si on supprime le mot-clé `public` ?

Exercice Dans la définition des différentes classes précédentes, programmer la méthode `str()` avec des `stringstream`

Exercice Dans la définition des différentes classes précédentes, programmer l'impression avec une fonction amie (`friend`) du module des `iostream`

```
friend ostream& operator << (ostream& o, const Personne& p) { … }
```

Prochain cours

- un bon tutorial C++: `http://www.programiz.com/cpp-programming`
- classes et objets en Python
- Java en tant que simplification de C++
- débuts en Java