Algo/Prog – DUT 1

0. Introduction
0. Appeler un sous-programme
0. Définition de procédures et fonctions
1. Conditionnelle (vu en TP)
0. Instructions complexe (boucles)
0. Variables, types
1. Représentation des données en mémoire
0. Boucles imbriquées et tableaux à doubles entrées
0. Pointeurs, etc
0. `struct`

0. Introduction
0. Appeler un sous-programme
0. Définition de procédures et fonctions
1. Conditionnelle (vu en TP)
0. Instructions complexe (boucles)
0. Variables, types
1. Représentation des données en mémoire
0. Boucles imbriquées et tableaux à doubles entrées
0. Pointeurs, etc
0. `struct`

Structure de données complexes

Comment relier plusieurs données de différents types ($\neq$ tableau) ?
Ex : nom + num sécurité sociale + solde.

- solution 1
  - plusieurs tableaux
  - association = même indice dans le tableaux
  - ex : tableau noms + tableau numéro S.S. + tableau soldes.
- solution 2
  - un seul tableau
  - cellule = noms + numéro S.S. + solde
    - ≪ tableau avec 3 cases de types différents ≫.
    - case 1 : nom, case 2 : num S.S., case 3 : solde etc.

Structure de données complexes

Comment relier plusieurs données de différents types ($\neq$ tableau) ?
Ex : nom + num sécurité sociale + solde.

- solution 1
  - plusieurs tableaux
  - association = même indice dans le tableaux
  - ex : tableau noms + tableau numéro S.S. + tableau soldes.
- solution 2

  taille fixe
  - un seul tableau
  - cellule = noms + numéro S.S. + solde
    - ★ « tableau avec 3 cases de types différents ».
    - ★ case 1 : nom, case 2 : num S.S., case 3 : solde etc.

Structure de données complexes

Comment relier plusieurs données de différents types ($\neq$ tableau) ?
Ex : nom + num sécurité sociale + solde.

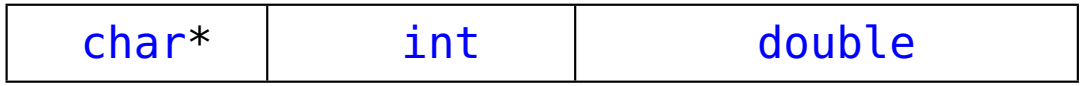
- solution 1
  - plusieurs tableaux
  - association = même indice dans le tableaux
  - ex : tableau noms + tableau numéro S.S. + tableau soldes.
- solution 2

  taille fixe
  - un seul tableau
  - cellule = noms + numéro S.S. + solde
    - ≪ tableau avec 3 cases de types différents ≫.
    - case 1 : nom, case 2 : num S.S., case 3 : solde etc.

  structure fixe

Tableaux hétérogènes

x =



- chaque case a son type propre ($\simeq$ objets dans les langages OO)
- records (enregistrements) dans les autres
- en C : `struct`

Principe

- nombre fixe de case
- désignées par des noms ~~(numéro)~~
- notation pointée : `t.nom` ~~(t[nom])~~

Tableaux hétérogènes

x =

x.nom	x.secu	x.solde
char*	int	double

- chaque case a son type propre ($\simeq$ objets dans les langages OO)
- records (enregistrements) dans les autres
- en C : `struct`

Principe

- nombre fixe de case
- désignées par des noms ~~(numéro)~~
- notation pointée : `t.nom` ~~(t[nom])~~

`struct` – déclaration, utilisation

```
struct { ... } var;
```

`struct` – déclaration, utilisation

```
struct { ... } var;
     int       n;
     char      c;
```

`struct` – déclaration, utilisation

```
struct { ... } var;
     int       n;
     char      c;
```

```
  struct {
    int nom1;
    char* nom2;
    double nom3; } x;
```

`struct` – déclaration, utilisation

```
struct { ... } var;
     int        n;
     char       c;
```

type

```
  struct {
    int nom1;
    char* nom2;
    double nom3; } x;
```

`struct` – déclaration, utilisation

```
struct { ... } var;
     int        n;
     char       c;
```

```
  struct {
    int nom1;
    char* nom2;          nom de variable
    double nom3; } x;
```

`struct` – déclaration, utilisation

```
struct { ... } var;
     int       n;
     char      c;
```

```
  struct {
    int nom1;
    char* nom2;
    double nom3; } x;
  x.nom1 = 12;
  x.nom2 = "toto";
  x.nom3 = 23.6;
  printf("%d , %s , %lf",x.nom1, x.nom2, x.nom3);
```

`struct` et `typedef`

Pour abréger : synonyme de type :

```
typedef struct { ... } nomtype;
```

`struct` et `typedef`

Pour abréger : synonyme de type :

```
typedef struct { ... } nomtype;
```

```
  typedef struct {
    int nom1;
    char* nom2;
    double nom3; } nomtype;

nomtype y;
  y.nom1 = 12;
```

`struct` et `typedef`

Pour abréger : synonyme de type :

```
typedef struct { ... } nomtype;
```

```
  typedef struct {
    int nom1;
    char* nom2;
    double nom3; } nomtype;
  type
nomtype y;
  y.nom1 = 12;
```

`struct *` l'outil ultime $\Rightarrow$ « objet »

```
typedef struct myobj { ... } *obj;
```

- `obj x = ...malloc...` « adresse d'une donnée complexe »
- `*x` donnée complexe : gros
- `x` : petit, bien pour paramètre de fonction `f(x)`. Pas de recopie de la donnée complexe.
- on accède quand on veut à la donnée complexe en faisant `(*x).nom`
- tellement pratiqué que
  - en C, notation dédiée : `x->nom`
  - notion centrale dans les langage objets (C++, Java).

DÉMO

`struct *` l'outil ultime $\Rightarrow$ « objet »

```
typedef struct myobj { ... } *obj;
```

- `obj x = ...malloc...` « adresse d'une donnée complexe »
- `*x` donnée complexe : gros
- `x` : petit, bien pour paramètre de fonction `f(x)`. Pas de recopie de la donnée complexe.
- on accède quand on veut à la donnée complexe en faisant `(*x).nom`
- tellement pratiqué que
  - en C, notation dédiée : `x->nom`
  - notion centrale dans les langage objets (C++, Java).

DÉMO

`struct` * l'outil ultime $\Rightarrow$ « objet »

pointeur

```
typedef struct myobj { ... } *obj;
```

- `obj x = ...malloc...` « adresse d'une donnée complexe »
- `*x` donnée complexe : gros
- `x` : petit, bien pour paramètre de fonction `f(x)`. Pas de recopie de la donnée complexe.
- on accède quand on veut à la donnée complexe en faisant `(*x).nom`
- tellement pratiqué que
  - en C, notation dédiée : `x->nom`
  - notion centrale dans les langage objets (C++, Java).

DÉMO