
- 1 Introduction
- 2 Premiers programmes
- 1 Définition de procédures et fonctions
- 2 Conditionnelle (vu en TP)
- 1 Instructions complexe (boucles)
- 1 Variables, types
- 2 Représentation des données en mémoire
- 1 Pointeurs, etc


Typage

```
int x;  
x = 8;  
... (x + 9) ...;  
... f(x) ...;          /* int f (int a) */
```

- chaque variable déclarée par un type T (`int`, `char`, ...)
- affectation : seulement par une valeur du type T
- opérateur seulement si T est le bon type
- fonction seulement si T est le bon type
- \Rightarrow Sécurité, pas d'erreur de type à l'exécution
- MAIS : conversion entre type (pratique mais source d'erreurs)

Coercions implicites

```
int x;  
x = 8.3;  
... (x + 9.2) ...;  
... f(12.5) ...;      /* int f (int a) */
```

- entorse aux règles de typage
- certains types sont convertibles entre eux (*coercion*)
-  conversion implicite malgré perte de précision éventuelle!

Coercions implicites

```
int x;  
x = 8.3;  
... (x + 9.2) ...;  
... f(12.5) ...;          /* int f (int a) */
```

affectation : conversion implicite vers le type de la variable

$x \leftarrow 8 \leftarrow 8.3$ opérateur : conversion des deux arguments vers un même type, plus grand ou égal aux deux types de départ

$x : \text{int}$ mais $9.2 : \text{float}$ donc on traduit tout vers float

$x = 8 \leftarrow 8.0 + 9.2 \rightsquigarrow 17.5$ fonction : arguments convertis vers le type attendu par la fonction

Notes

Coercions implicites

```
int x;  
x = 8.9 + 9.2;
```

- ?
- $x \leftarrow 18$

Notes

Coercions implicites

```
int x=8;  
double y = 2.3;  
x = x/y;
```

- ?
- $x \leftarrow 3$

Notes

Coercions implicites

```
int x=8.3;  
double y = 2.3;  
x = x/y;
```

- ?
- $x \leftarrow 3$

Notes

Coercion : priorités

```
int n = 8;
double x = 2.6;
y = n / 3 + x;
printf ("n = %d , x = %fd, y = %fd\n",n,x,y);
```

- $n = 8$, $x = 2.600000d$, $y = 4.600000d$
- $n / 3 \rightsquigarrow 2$ (int)
- $2 + 2.6 \rightsquigarrow 4.6$

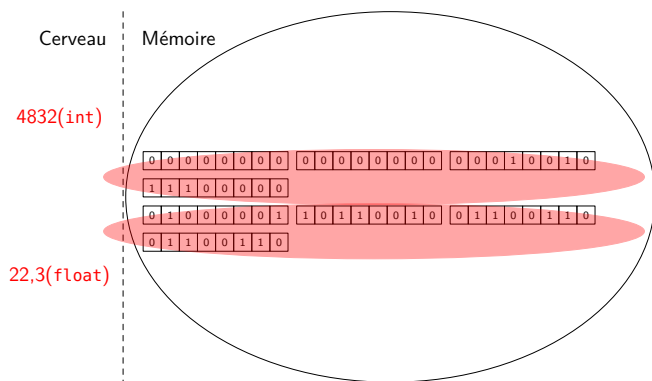
Notes

Donnée vues \neq Données mémoires

- Données pensées par l'utilisateur :
 - ▶ entiers, réels
 - ▶ textes
 - ▶ « grille » du morpion
- Données dans la mémoire de l'ordinateur :
 - ▶ des octets
 - ▶ des octets
 - ▶ des octets

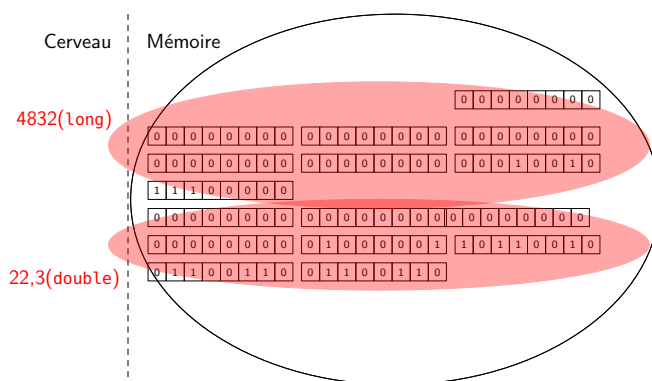
Notes

Représentation Mémoire – Les nombres



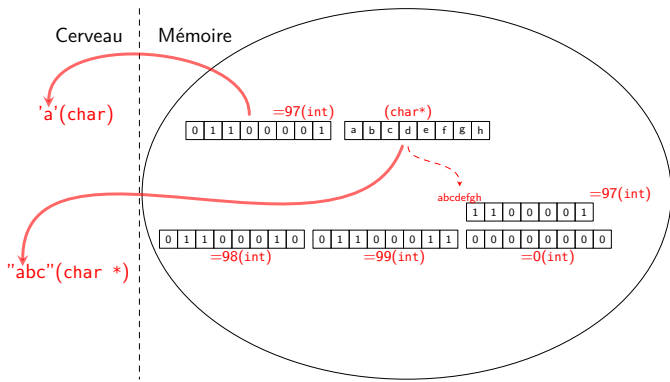
Notes

Représentation Mémoire – Les nombres



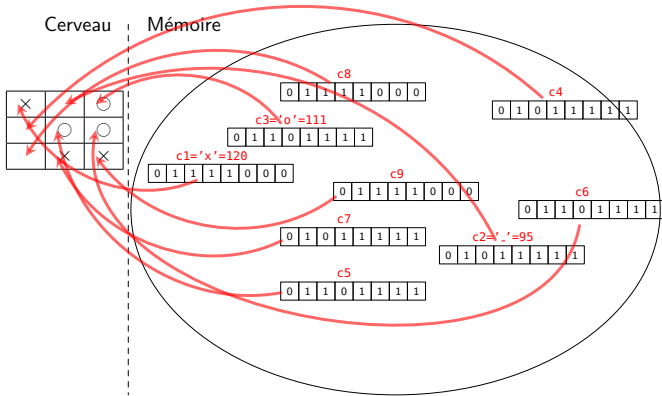
Notes

Représentation Mémoire – Les caractères



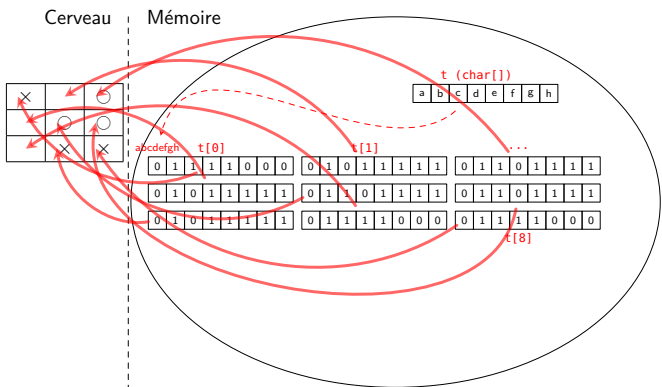
Notes

Représentation Mémoire – La grille du morpion



Notes

Représentation Mémoire – La grille du morpion



Notes

Représentation Mémoire

- l'important : cohérence entre représentation mémoire et algorithmes
- programmer = choisir une représentation mémoire pertinente
- + concevoir les algorithmes pour la manipuler

Notes

Représentation Mémoire

Critère de pertinence	c1...c9	char t[9]
Correcte (fidèle)	OK	OK
Complète (toutes les informations)	OK	OK
Pratique (facile à manipuler)	Bof	OK
Efficace (en taille et en calcul)	OK	OK

- une seule variable, manipulation du numéro de case via l'indice
- meilleur passage à l'échelle (30×30...)
- futures évolutions (puissance 4, bataille navale etc)
- fonctions utiles pour d'autres jeux (`voisinDroite`, `voisinHaut`, etc)

Notes

Notes

Notes

Notes
