Initiation à l'algorithmique
Les pointeurs

Mohamed MESSABIHI

mohamed.messabihi@gmail.com

Université de Tlemcen
Département d'informatique
1ère année MI

https://sites.google.com/site/informatiquemessabihi/

Introduction

Exercice

Écrire une fonction à laquelle on envoie une durée exprimée en minutes. Celle-ci renverrait le nombre d'heures et minutes correspondantes :

- si on envoie 45, la fonction renvoie 0 heure et 45 minutes ;
- si on envoie 60, la fonction renvoie 1 heure et 0 minutes ;
- si on envoie 90, la fonction renvoie 1 heure et 30 minutes.

Problèmes :

- En effet, on ne peut renvoyer qu'une valeur par fonction !
- On peut utiliser des variables globales mais cette pratique est fortement déconseillée.

Introduction

Exercice

Écrire une fonction à laquelle on envoie une durée exprimée en minutes. Celle-ci renverrait le nombre d'heures et minutes correspondantes :

- si on envoie 45, la fonction renvoie 0 heure et 45 minutes ;
- si on envoie 60, la fonction renvoie 1 heure et 0 minutes ;
- si on envoie 90, la fonction renvoie 1 heure et 30 minutes.

Problèmes :

- En effet, on ne peut renvoyer qu'une valeur par fonction !
- On peut utiliser des variables globales mais cette pratique est fortement déconseillée.

Introduction

Exercice

Écrire une fonction à laquelle on envoie une durée exprimée en minutes. Celle-ci renverrait le nombre d'heures et minutes correspondantes :

- si on envoie 45, la fonction renvoie 0 heure et 45 minutes ;
- si on envoie 60, la fonction renvoie 1 heure et 0 minutes ;
- si on envoie 90, la fonction renvoie 1 heure et 30 minutes.

Problèmes :

- En effet, on ne peut renvoyer qu'une valeur par fonction !
- On peut utiliser des variables globales mais cette pratique est fortement déconseillée.

Introduction

Exercice

Écrire une fonction à laquelle on envoie une durée exprimée en minutes. Celle-ci renverrait le nombre d'heures et minutes correspondantes :

- si on envoie 45, la fonction renvoie 0 heure et 45 minutes ;
- si on envoie 60, la fonction renvoie 1 heure et 0 minutes ;
- si on envoie 90, la fonction renvoie 1 heure et 30 minutes.

Problèmes :

- En effet, on ne peut renvoyer qu'une valeur par fonction !
- On peut utiliser des variables globales mais cette pratique est fortement déconseillée.

Introduction

Exercice

Écrire une fonction à laquelle on envoie une durée exprimée en minutes. Celle-ci renverrait le nombre d'heures et minutes correspondantes :

- si on envoie 45, la fonction renvoie 0 heure et 45 minutes ;
- si on envoie 60, la fonction renvoie 1 heure et 0 minutes ;
- si on envoie 90, la fonction renvoie 1 heure et 30 minutes.

Problèmes :

- En effet, on ne peut renvoyer qu'une valeur par fonction !
- On peut utiliser des variables globales mais cette pratique est fortement déconseillée.

Introduction

Exercice

Écrire une fonction à laquelle on envoie une durée exprimée en minutes. Celle-ci renverrait le nombre d'heures et minutes correspondantes :

- si on envoie 45, la fonction renvoie 0 heure et 45 minutes ;
- si on envoie 60, la fonction renvoie 1 heure et 0 minutes ;
- si on envoie 90, la fonction renvoie 1 heure et 30 minutes.

Problèmes :

- En effet, on ne peut renvoyer qu'une valeur par fonction !
- On peut utiliser des variables globales mais cette pratique est fortement déconseillée.

Solution : notion de pointeur

On doit donc apprendre à se servir de la notion de pointeur...

Adresse	Valeur
0	145
1	3.8028322
2	0.827551
3	3901930
...	...
3 448 765 900 126 (et des poussières)	940.5118

Adresse vs. Valeur

Quand vous créez une variable age de type int par exemple, en tapant :

```
int age = 10;
```

1. votre programme demande au système d'exploitation (Windows, par exemple) la permission d'utiliser un peu de mémoire.
2. Le système d'exploitation répond en indiquant à quelle adresse en mémoire il vous laisse le droit d'inscrire votre nombre.
3. La valeur 10 est inscrite quelque part en mémoire, disons par exemple à l'adresse 4655.
4. Ensuite, le compilateur remplace le mot age dans votre programme par l'adresse 4655 à l'exécution
5. Donc, l'ordinateur se rendra toujours à l'adresse 4655 pour récupérer la valeur de la variable « age ».

Adresse vs. Valeur

Quand vous créez une variable age de type int par exemple, en tapant :

```
int age = 10;
```

1. votre programme demande au système d'exploitation (Windows, par exemple) la permission d'utiliser un peu de mémoire.
2. Le système d'exploitation répond en indiquant à quelle adresse en mémoire il vous laisse le droit d'inscrire votre nombre.
3. La valeur 10 est inscrite quelque part en mémoire, disons par exemple à l'adresse 4655.
4. Ensuite, le compilateur remplace le mot age dans votre programme par l'adresse 4655 à l'exécution
5. Donc, l'ordinateur se rendra toujours à l'adresse 4655 pour récupérer la valeur de la variable « age ».

Adresse vs. Valeur

Quand vous créez une variable age de type int par exemple, en tapant :

```
int age = 10;
```

1. votre programme demande au système d'exploitation (Windows, par exemple) la permission d'utiliser un peu de mémoire.
2. Le système d'exploitation répond en indiquant à quelle adresse en mémoire il vous laisse le droit d'inscrire votre nombre.
3. La valeur 10 est inscrite quelque part en mémoire, disons par exemple à l'adresse 4655.
4. Ensuite, le compilateur remplace le mot age dans votre programme par l'adresse 4655 à l'exécution
5. Donc, l'ordinateur se rendra toujours à l'adresse 4655 pour récupérer la valeur de la variable « age ».

Adresse vs. Valeur

Quand vous créez une variable age de type int par exemple, en tapant :

```
int age = 10;
```

1. votre programme demande au système d'exploitation (Windows, par exemple) la permission d'utiliser un peu de mémoire.
2. Le système d'exploitation répond en indiquant à quelle adresse en mémoire il vous laisse le droit d'inscrire votre nombre.
3. La valeur 10 est inscrite quelque part en mémoire, disons par exemple à l'adresse 4655.
4. Ensuite, le compilateur remplace le mot age dans votre programme par l'adresse 4655 à l'exécution
5. Donc, l'ordinateur se rendra toujours à l'adresse 4655 pour récupérer la valeur de la variable « age ».

Adresse vs. Valeur

Quand vous créez une variable age de type int par exemple, en tapant :

```
int age = 10;
```

1. votre programme demande au système d'exploitation (Windows, par exemple) la permission d'utiliser un peu de mémoire.
2. Le système d'exploitation répond en indiquant à quelle adresse en mémoire il vous laisse le droit d'inscrire votre nombre.
3. La valeur 10 est inscrite quelque part en mémoire, disons par exemple à l'adresse 4655.
4. Ensuite, le compilateur remplace le mot age dans votre programme par l'adresse 4655 à l'exécution
5. Donc, l'ordinateur se rendra toujours à l'adresse 4655 pour récupérer la valeur de la variable « age ».

Adresse vs. valeur

Comment récupérer l'adresse d'une variable ?

Exemple :

```c
int age = 10;
printf("La variable age vaut : %d", age);
printf("L'adresse de la variable age est : %p", &age);
```

```
La variable age vaut : 10
L'adresse de la variable age est : 0028FF1C
```

Donc à retenir :

1. age : désigne la valeur de la variable ;
2. & age : désigne l'adresse de la variable.

Adresse vs. valeur

Comment récupérer l'adresse d'une variable ?

Exemple :

```c
int age = 10;
printf("La variable age vaut : %d", age);
printf("L'adresse de la variable age est : %p", &age);
```

```
La variable age vaut : 10
L'adresse de la variable age est : 0028FF1C
```

Donc à retenir :

1. age : désigne la valeur de la variable ;
2. & age : désigne l'adresse de la variable.

Adresse vs. valeur

Comment récupérer l'adresse d'une variable ?

Exemple :

```c
int age = 10;
printf("La variable age vaut : %d", age);
printf("L'adresse de la variable age est : %p", &age);
```

```
La variable age vaut : 10
L'adresse de la variable age est : 0028FF1C
```

Donc à retenir :

1. age : désigne la valeur de la variable ;
2. & age : désigne l'adresse de la variable.

C'est quoi un pointeur ?

- Jusqu'ici, nous avons uniquement créé des variables faites pour contenir des nombres.
- Maintenant, nous allons apprendre à créer des variables faites pour contenir des adresses
- Ce sont justement ce qu'on appelle des pointeurs.

Déclaration d'un pointeur :

```
int *monPointeur;
```

- Pour créer une variable de type pointeur, on doit rajouter le symbole * devant le nom de la variable.
- Notez qu'on peut aussi écrire int* monPointeur ;

C'est quoi un pointeur ?

- Jusqu'ici, nous avons uniquement créé des variables faites pour contenir des nombres.
- Maintenant, nous allons apprendre à créer des variables faites pour contenir des adresses
- Ce sont justement ce qu'on appelle des pointeurs.

Déclaration d'un pointeur :

```
int *monPointeur;
```

- Pour créer une variable de type pointeur, on doit rajouter le symbole * devant le nom de la variable.
- Notez qu'on peut aussi écrire int* monPointeur ;

C'est quoi un pointeur ?

- Jusqu'ici, nous avons uniquement créé des variables faites pour contenir des nombres.
- Maintenant, nous allons apprendre à créer des variables faites pour contenir des adresses
- Ce sont justement ce qu'on appelle des pointeurs.

Déclaration d'un pointeur :

```
int *monPointeur;
```

- Pour créer une variable de type pointeur, on doit rajouter le symbole * devant le nom de la variable.
- Notez qu'on peut aussi écrire int* monPointeur ;

C'est quoi un pointeur ?

- Jusqu'ici, nous avons uniquement créé des variables faites pour contenir des nombres.
- Maintenant, nous allons apprendre à créer des variables faites pour contenir des adresses
- Ce sont justement ce qu'on appelle des pointeurs.

Déclaration d'un pointeur :

```
int *monPointeur;
```

- Pour créer une variable de type pointeur, on doit rajouter le symbole * devant le nom de la variable.
- Notez qu'on peut aussi écrire int* monPointeur ;

C'est quoi un pointeur ?

- Jusqu'ici, nous avons uniquement créé des variables faites pour contenir des nombres.
- Maintenant, nous allons apprendre à créer des variables faites pour contenir des adresses
- Ce sont justement ce qu'on appelle des pointeurs.

Déclaration d'un pointeur :

```
int *monPointeur;
```

- Pour créer une variable de type pointeur, on doit rajouter le symbole * devant le nom de la variable.
- Notez qu'on peut aussi écrire int* monPointeur ;

C'est quoi un pointeur ?

- Jusqu'ici, nous avons uniquement créé des variables faites pour contenir des nombres.
- Maintenant, nous allons apprendre à créer des variables faites pour contenir des adresses
- Ce sont justement ce qu'on appelle des pointeurs.

Déclaration d'un pointeur :

```
int *monPointeur;
```

- Pour créer une variable de type pointeur, on doit rajouter le symbole * devant le nom de la variable.
- Notez qu'on peut aussi écrire int* monPointeur ;

Initialiser et affecter une adresse à un pointeur

- Pour initialiser un pointeur, c'est-à-dire lui donner une valeur par défaut, on n'utilise généralement pas le nombre 0 mais le mot-clé NULL (veillez à l'écrire en majuscules)

Exemple

```c
int *monPointeur = NULL;

int age = 10;
int *pointeurSurAge = &age;
```

- La première ligne réserve une case en mémoire pour contenir une adresse mais que cette case ne contient aucune adresse pour le moment.
- La seconde ligne signifie : « Créer une variable de type int dont la valeur vaut 10 ».
- La dernière ligne signifie : « Créer une variable de type pointeur dont la valeur vaut l'**adresse** de la variable age ».

Initialiser et affecter une adresse à un pointeur

- Pour initialiser un pointeur, c'est-à-dire lui donner une valeur par défaut, on n'utilise généralement pas le nombre 0 mais le mot-clé NULL (veillez à l'écrire en majuscules)

Exemple

```
int *monPointeur = NULL;

int age = 10;
int *pointeurSurAge = &age;
```

- La première ligne réserve une case en mémoire pour contenir une adresse mais que cette case ne contient aucune adresse pour le moment.
- La seconde ligne signifie : « Créer une variable de type int dont la valeur vaut 10 ».
- La dernière ligne signifie : « Créer une variable de type pointeur dont la valeur vaut l'**adresse** de la variable age ».

Initialiser et affecter une adresse à un pointeur

- Pour initialiser un pointeur, c'est-à-dire lui donner une valeur par défaut, on n'utilise généralement pas le nombre 0 mais le mot-clé NULL (veillez à l'écrire en majuscules)

Exemple

```
int *monPointeur = NULL;

int age = 10;
int *pointeurSurAge = &age;
```

- La première ligne réserve une case en mémoire pour contenir une adresse mais que cette case ne contient aucune adresse pour le moment.
- La seconde ligne signifie : « Créer une variable de type int dont la valeur vaut 10 ».
- La dernière ligne signifie : « Créer une variable de type pointeur dont la valeur vaut l'**adresse** de la variable age ».

Initialiser et affecter une adresse à un pointeur

- Pour initialiser un pointeur, c'est-à-dire lui donner une valeur par défaut, on n'utilise généralement pas le nombre 0 mais le mot-clé NULL (veillez à l'écrire en majuscules)

Exemple

```
int *monPointeur = NULL;

int age = 10;
int *pointeurSurAge = &age;
```

- La première ligne réserve une case en mémoire pour contenir une adresse mais que cette case ne contient aucune adresse pour le moment.
- La seconde ligne signifie : « Créer une variable de type int dont la valeur vaut 10 ».
- La dernière ligne signifie : « Créer une variable de type pointeur dont la valeur vaut l'**adresse** de la variable age ».

Initialiser et affecter une adresse à un pointeur

- Pour initialiser un pointeur, c'est-à-dire lui donner une valeur par défaut, on n'utilise généralement pas le nombre 0 mais le mot-clé NULL (veillez à l'écrire en majuscules)

Exemple

```
int *monPointeur = NULL;

int age = 10;
int *pointeurSurAge = &age;
```

- La première ligne réserve une case en mémoire pour contenir une adresse mais que cette case ne contient aucune adresse pour le moment.
- La seconde ligne signifie : « Créer une variable de type int dont la valeur vaut 10 ».
- La dernière ligne signifie : « Créer une variable de type pointeur dont la valeur vaut l'**adresse** de la variable age ».

Les pointeurs ont-ils un type ?

- Il n'y a pas de type « pointeur » comme il y a un type int et un type double. On n'écrit donc pas pointeur pointeurSurAge ;
- On utilise le symbole *, mais on continue à indiquer quel est le type de la variable dont le pointeur va contenir l'adresse.

Exemple :

```
int age = 10;
int *pointeurSurAge = &age;
```

- Comme le pointeur pointeurSurAge va contenir l'adresse de la variable age (qui est de type int), alors le pointeur doit être de type int*.
- Si la variable age avait été de type double, alors on aurait dû écrire double *monPointeur.

Les pointeurs ont-ils un type ?

- Il n'y a pas de type « pointeur » comme il y a un type int et un type double. On n'écrit donc pas pointeur pointeurSurAge ;
- On utilise le symbole *, mais on continue à indiquer quel est le type de la variable dont le pointeur va contenir l'adresse.

Exemple :

```
int age = 10;
int *pointeurSurAge = &age;
```

- Comme le pointeur pointeurSurAge va contenir l'adresse de la variable age (qui est de type int), alors le pointeur doit être de type int*.
- Si la variable age avait été de type double, alors on aurait dû écrire double *monPointeur.

Les pointeurs ont-ils un type ?

- Il n'y a pas de type « pointeur » comme il y a un type int et un type double. On n'écrit donc pas pointeur pointeurSurAge ;
- On utilise le symbole *, mais on continue à indiquer quel est le type de la variable dont le pointeur va contenir l'adresse.

Exemple :

```
int age = 10;
int *pointeurSurAge = &age;
```

- Comme le pointeur pointeurSurAge va contenir l'adresse de la variable age (qui est de type int), alors le pointeur doit être de type int*.
- Si la variable age avait été de type double, alors on aurait dû écrire double *monPointeur.

Les pointeurs ont-ils un type ?

- Il n'y a pas de type « pointeur » comme il y a un type int et un type double. On n'écrit donc pas pointeur pointeurSurAge ;
- On utilise le symbole *, mais on continue à indiquer quel est le type de la variable dont le pointeur va contenir l'adresse.

Exemple :

```
int age = 10;
int *pointeurSurAge = &age;
```

- Comme le pointeur pointeurSurAge va contenir l'adresse de la variable age (qui est de type int), alors le pointeur doit être de type int*.
- Si la variable age avait été de type double, alors on aurait dû écrire double *monPointeur.

Les pointeurs ont-ils un type ?

- Il n'y a pas de type « pointeur » comme il y a un type int et un type double. On n'écrit donc pas pointeur pointeurSurAge ;
- On utilise le symbole *, mais on continue à indiquer quel est le type de la variable dont le pointeur va contenir l'adresse.

Exemple :

```
int age = 10;
int *pointeurSurAge = &age;
```

- Comme le pointeur pointeurSurAge va contenir l'adresse de la variable age (qui est de type int), alors le pointeur doit être de type int*.
- Si la variable age avait été de type double, alors on aurait dû écrire double *monPointeur.

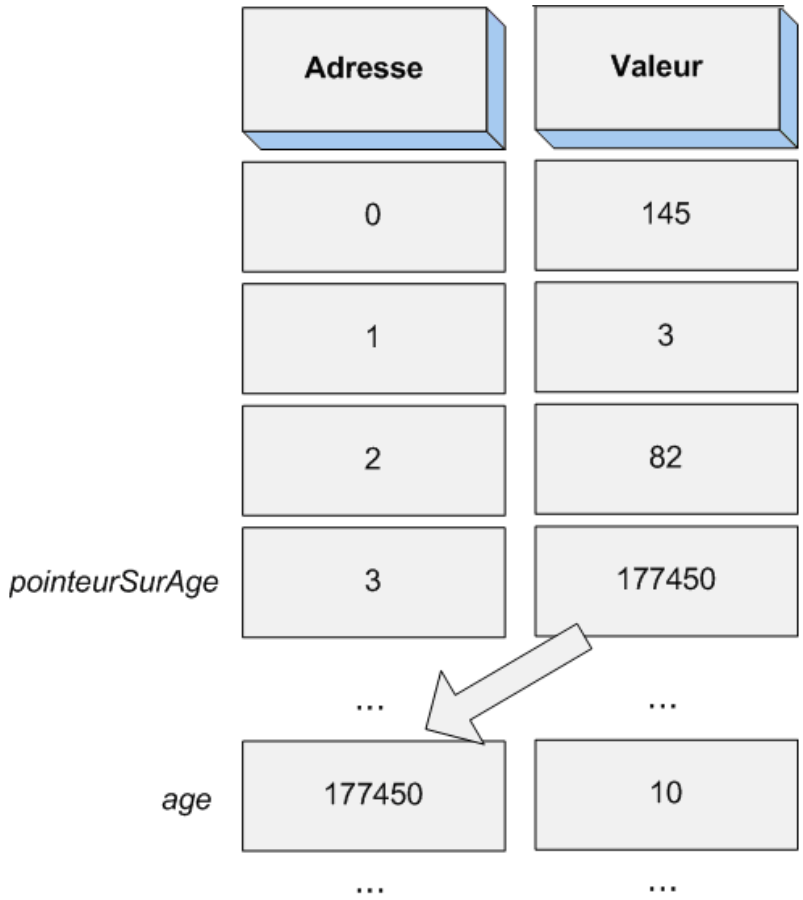
Schéma explicatif

Exemple :

```
int age = 10;
int *pointeurSurAge = &age;
```

Vocabulaire :

On dit que le pointeur **pointeurSurAge** pointe sur la variable **age**.



La valeur d'un pointeur

Exemple :

```
int age = 10;
int *pointeurSurAge = &age;

printf("%d", pointeurSurAge);
printf("%d", *pointeurSurAge);
printf("%d", &pointeurSurAge);
```

- Le premier appel de printf affiche la valeur de pointeurSurAge, et sa valeur c'est l'adresse de la variable age (177450).
- Le second appel de printf affiche la valeur de la variable se trouvant à l'adresse indiquée dans pointeurSurAge.
- Il faut donc placer le symbole * devant le nom du pointeur pour récupérer la valeur de la variable se trouvant à l'adresse indiquée dans un pointeur.
- ler dernier appel de printf affiche l'adresse à laquelle se trouve le pointeur (ici, c'est 3).

La valeur d'un pointeur

Exemple :

```
int age = 10;
int *pointeurSurAge = &age;

printf("%d", pointeurSurAge);
printf("%d", *pointeurSurAge);
printf("%d", &pointeurSurAge);
```

- Le premier appel de printf affiche la valeur de pointeurSurAge, et sa valeur c'est l'adresse de la variable age (177450).
- Le second appel de printf affiche la valeur de la variable se trouvant à l'adresse indiquée dans pointeurSurAge.
- Il faut donc placer le symbole * devant le nom du pointeur pour récupérer la valeur de la variable se trouvant à l'adresse indiquée dans un pointeur.
- ler dernier appel de printf affiche l'adresse à laquelle se trouve le pointeur (ici, c'est 3).

La valeur d'un pointeur

Exemple :

```c
int age = 10;
int *pointeurSurAge = &age;

printf("%d", pointeurSurAge);
printf("%d", *pointeurSurAge);
printf("%d", &pointeurSurAge);
```

- Le premier appel de printf affiche la valeur de pointeurSurAge, et sa valeur c'est l'adresse de la variable age (177450).
- Le second appel de printf affiche la valeur de la variable se trouvant à l'adresse indiquée dans pointeurSurAge.
- Il faut donc placer le symbole * devant le nom du pointeur pour récupérer la valeur de la variable se trouvant à l'adresse indiquée dans un pointeur.
- ler dernier appel de printf affiche l'adresse à laquelle se trouve le pointeur (ici, c'est 3).

La valeur d'un pointeur

Exemple :

```
int age = 10;
int *pointeurSurAge = &age;

printf("%d", pointeurSurAge);
printf("%d", *pointeurSurAge);
printf("%d", &pointeurSurAge);
```

- Le premier appel de printf affiche la valeur de pointeurSurAge, et sa valeur c'est l'adresse de la variable age (177450).
- Le second appel de printf affiche la valeur de la variable se trouvant à l'adresse indiquée dans pointeurSurAge.
- Il faut donc placer le symbole * devant le nom du pointeur pour récupérer la valeur de la variable se trouvant à l'adresse indiquée dans un pointeur.
- ler dernier appel de printf affiche l'adresse à laquelle se trouve le pointeur (ici, c'est 3).

La valeur d'un pointeur

Exemple :

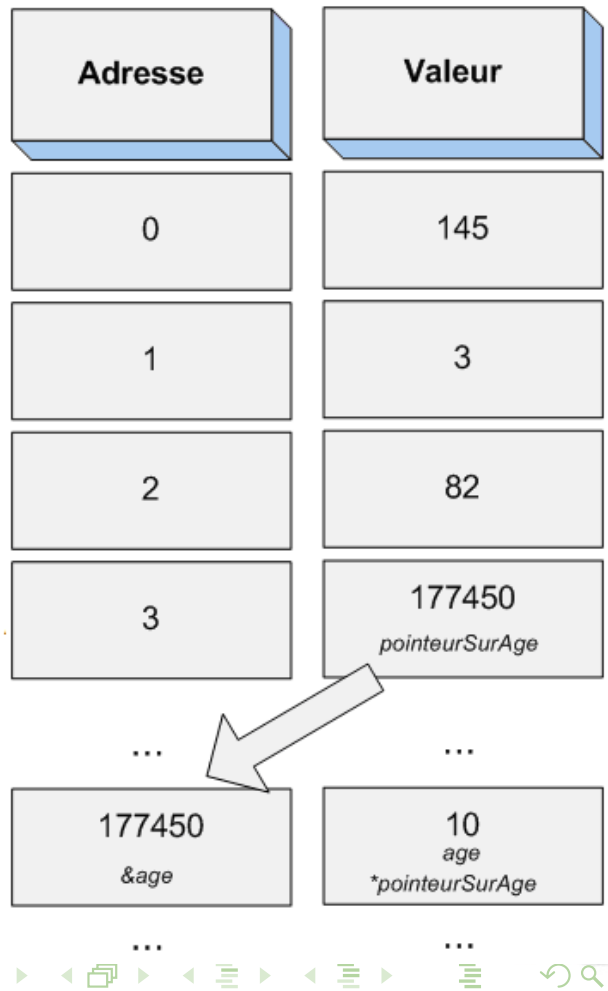
```
int age = 10;
int *pointeurSurAge = &age;

printf("%d", pointeurSurAge);
printf("%d", *pointeurSurAge);
printf("%d", &pointeurSurAge);
```

- Le premier appel de printf affiche la valeur de pointeurSurAge, et sa valeur c'est l'adresse de la variable age (177450).
- Le second appel de printf affiche la valeur de la variable se trouvant à l'adresse indiquée dans pointeurSurAge.
- Il faut donc placer le symbole * devant le nom du pointeur pour récupérer la valeur de la variable se trouvant à l'adresse indiquée dans un pointeur.
- ler dernier appel de printf affiche l'adresse à laquelle se trouve le pointeur (ici, c'est 3).

À retenir

Les pointeurs et les noms de variables ont le même rôle : Ils donnent accès à un emplacement dans la mémoire interne de l'ordinateur. Il faut quand même bien faire la différence :

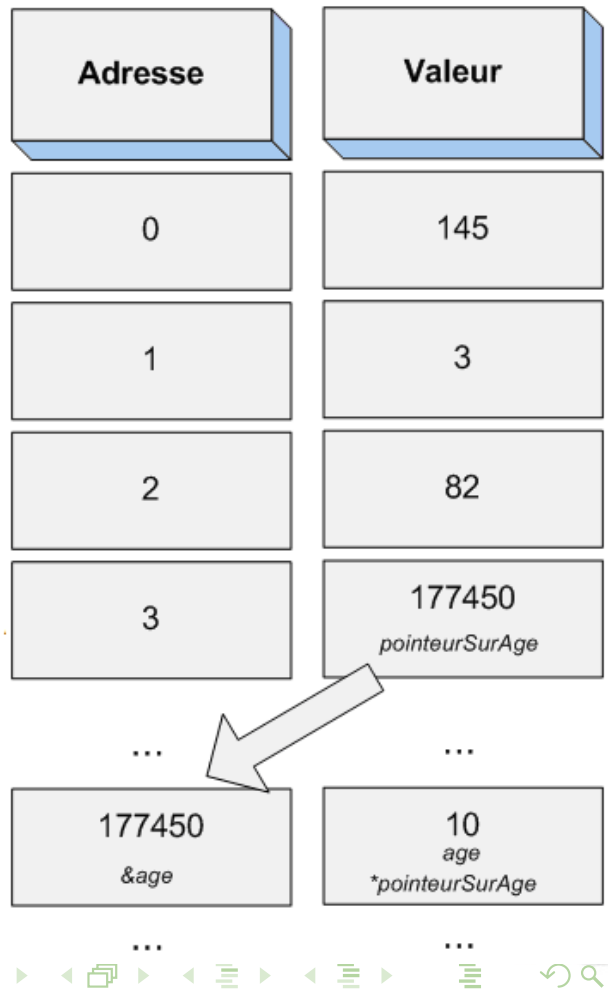
- Le nom d'une variable reste toujours lié à la même adresse.
  - **age** signifie : « Je veux la valeur de la variable age »,
  - **& age** signifie : « Je veux l'adresse à laquelle se trouve la variable age » ;
- Un pointeur est une variable qui peut **pointer** sur différentes adresses.
  - pointeurSurAge signifie : « Je veux la valeur de pointeurSurAge » (cette valeur étant une adresse),
  - *pointeurSurAge signifie : « Je veux la valeur de la variable pointée à l'adresse contenue dans pointeurSurAge »



À retenir

Les pointeurs et les noms de variables ont le même rôle : Ils donnent accès à un emplacement dans la mémoire interne de l'ordinateur. Il faut quand même bien faire la différence :

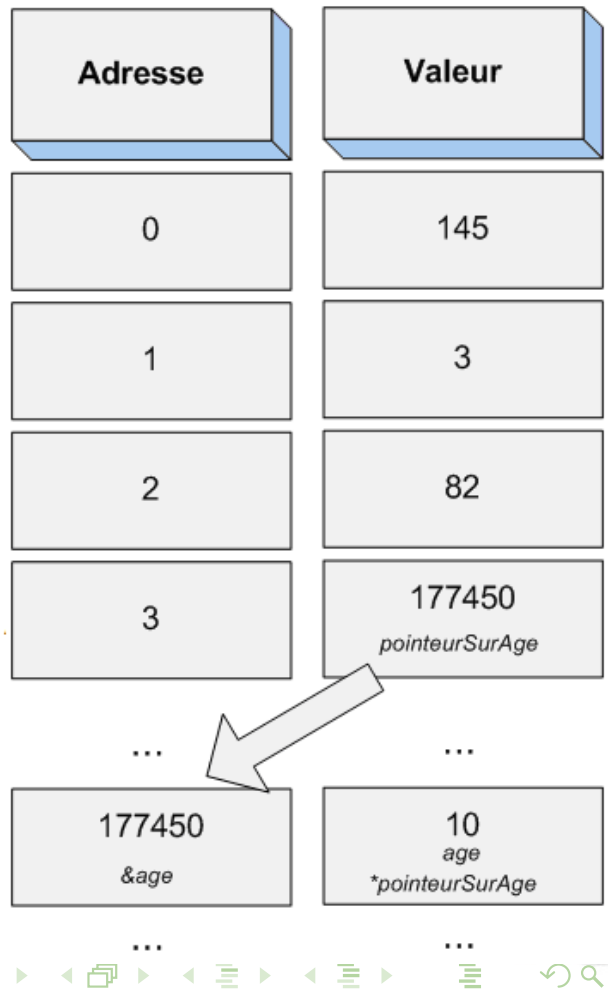
- Le nom d'une variable reste toujours lié à la même adresse.
  - **age** signifie : « Je veux la valeur de la variable age »,
  - **& age** signifie : « Je veux l'adresse à laquelle se trouve la variable age » ;
- Un pointeur est une variable qui peut **pointer** sur différentes adresses.
  - pointeurSurAge signifie : « Je veux la valeur de pointeurSurAge » (cette valeur étant une adresse),
  - *pointeurSurAge signifie : « Je veux la valeur de la variable qui se trouve à l'adresse contenue dans pointeurSurAge ».



À retenir

Les pointeurs et les noms de variables ont le même rôle : Ils donnent accès à un emplacement dans la mémoire interne de l'ordinateur. Il faut quand même bien faire la différence :

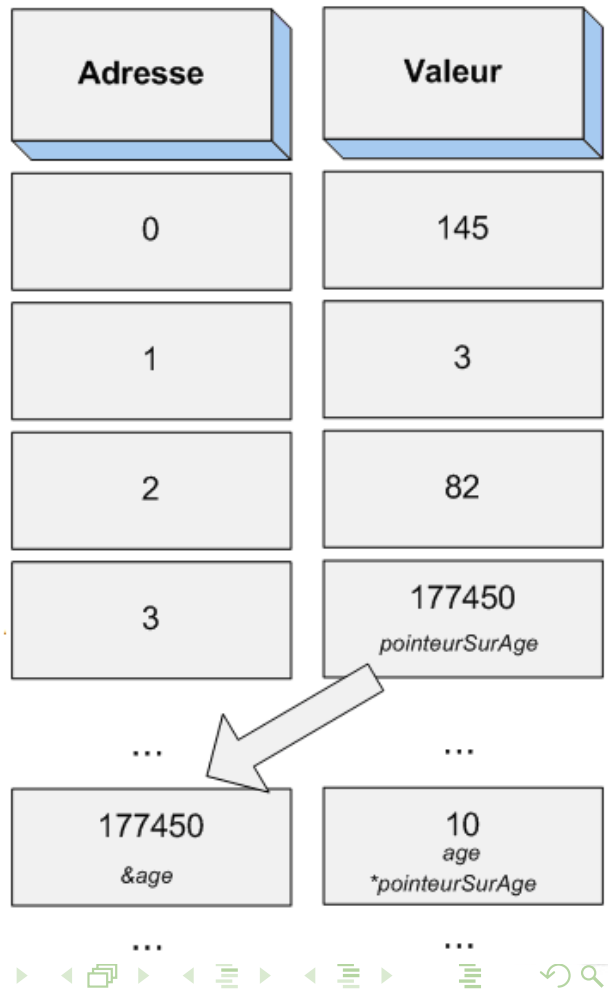
- Le nom d'une variable reste toujours lié à la même adresse.
  - **age** signifie : « Je veux la valeur de la variable age »,
  - **& age** signifie : « Je veux l'adresse à laquelle se trouve la variable age » ;
- Un pointeur est une variable qui peut **pointer** sur différentes adresses.
  - pointeurSurAge signifie : « Je veux la valeur de pointeurSurAge » (cette valeur étant une adresse),
  - *pointeurSurAge signifie : « Je veux la valeur de la variable qui se trouve à l'adresse contenue dans pointeurSurAge ».



À retenir

Les pointeurs et les noms de variables ont le même rôle : Ils donnent accès à un emplacement dans la mémoire interne de l'ordinateur. Il faut quand même bien faire la différence :

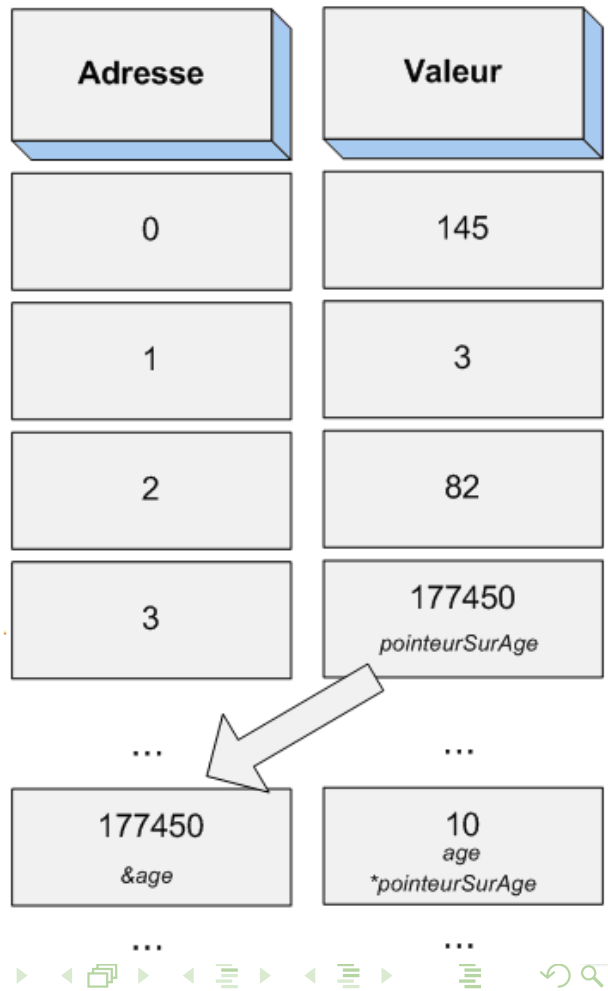
- Le nom d'une variable reste toujours lié à la même adresse.
  - **age** signifie : « Je veux la valeur de la variable age »,
  - **& age** signifie : « Je veux l'adresse à laquelle se trouve la variable age » ;
- Un pointeur est une variable qui peut **pointer** sur différentes adresses.
  - **pointeurSurAge** signifie : « Je veux la valeur de pointeurSurAge » (cette valeur étant une adresse),
  - ***pointeurSurAge** signifie : « Je veux la valeur de la variable qui se trouve à l'adresse contenue dans pointeurSurAge ».



À retenir

Les pointeurs et les noms de variables ont le même rôle : Ils donnent accès à un emplacement dans la mémoire interne de l'ordinateur. Il faut quand même bien faire la différence :

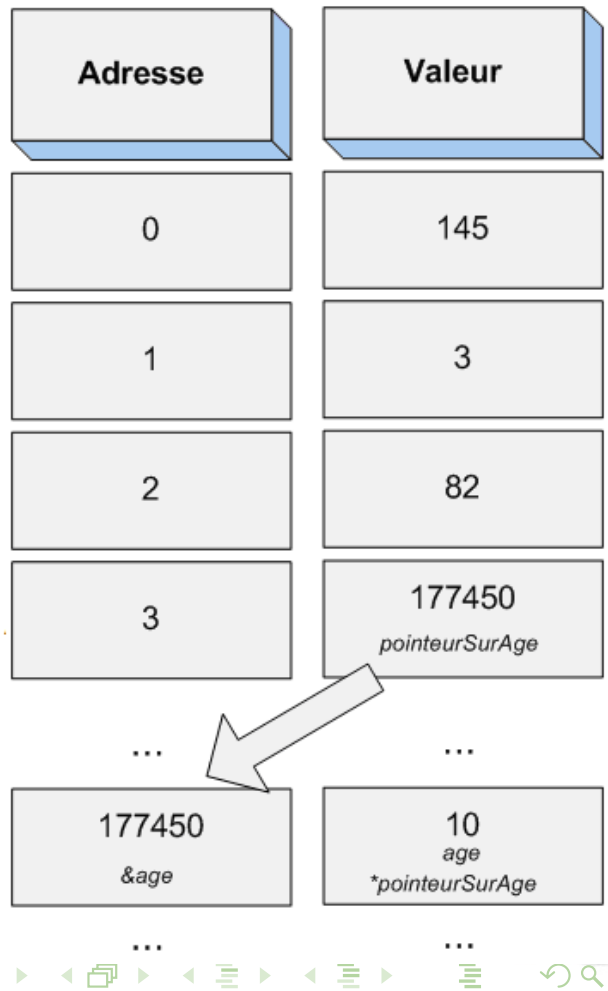
- Le nom d'une variable reste toujours lié à la même adresse.
  - **age** signifie : « Je veux la valeur de la variable age »,
  - **& age** signifie : « Je veux l'adresse à laquelle se trouve la variable age » ;
- Un pointeur est une variable qui peut **pointer** sur différentes adresses.
  - **pointeurSurAge** signifie : « Je veux la valeur de pointeurSurAge » (cette valeur étant une adresse),
  - ***pointeurSurAge** signifie : « Je veux la valeur de la variable qui se trouve à l'adresse contenue dans pointeurSurAge ».



À retenir

Les pointeurs et les noms de variables ont le même rôle : Ils donnent accès à un emplacement dans la mémoire interne de l'ordinateur. Il faut quand même bien faire la différence :

- Le nom d'une variable reste toujours lié à la même adresse.
  - **age** signifie : « Je veux la valeur de la variable age »,
  - **& age** signifie : « Je veux l'adresse à laquelle se trouve la variable age » ;
- Un pointeur est une variable qui peut **pointer** sur différentes adresses.
  - **pointeurSurAge** signifie : « Je veux la valeur de pointeurSurAge » (cette valeur étant une adresse),
  - ***pointeurSurAge** signifie : « Je veux la valeur de la variable qui se trouve à l'adresse contenue dans pointeurSurAge ».



Déclaration vs. Utilisation d'un pointeur

Attention :

Ne pas confondre les différentes significations de l'étoile !

1. Lorsque on **déclare** un pointeur, l'étoile sert juste à indiquer qu'on veut créer un pointeur : int *pointeurSurAge ;.
2. En revanche, lorsqu'ensuite on **utilise** le pointeur en écrivant printf("%d", *pointeurSurAge) ;, cela ne signifie pas « on veut créer un pointeur » mais : « on veut la valeur de la variable sur laquelle pointe le pointeurSurAge ».

Exemple :

```
int age = 10;
int *pointeurSurAge = &age;      //declaration d'un pointeur

*pointeurSurAge += 1;            //utilisation du pointeur
printf("%d", *pointeurSurAge);   //utilisation du pointeur
```

Passage par adresse des paramètres d'une fonction

Le gros intérêt des pointeurs (mais ce n'est pas le seul) est qu'on peut les envoyer à des fonctions pour qu'elles modifient directement une variable en mémoire, et non une copie comme on l'a déjà vu auparavant.

Exemple :

```
void triplePointeur(int *pointeurSurNombre);
int main()
{
    int nombre = 5;

    triplePointeur(&nombre); // On envoie l'adresse de
        nombre a la fonction
    printf("%d", nombre); // On affiche la variable nombre.

    return 0;
}
void triplePointeur(int *pointeurSurNombre)
{
    *pointeurSurNombre *= 3;
}
```

Passage par adresse des paramètres d'une fonctions

En exécutant le programme du slide précédent, voici ce qu'il se passe dans l'ordre, en partant du début du main :

1. une variable **nombre** est créée dans le main. On lui affecte la valeur 5.
2. on appelle la fonction **triplePointeur**. On lui envoie en paramètre l'adresse de notre variable nombre ;
3. la fonction **triplePointeur** reçoit cette adresse dans **pointeurSurNombre**. À l'intérieur de la fonction **triplePointeur**, on a donc un pointeur **pointeurSurNombre** qui contient l'adresse de la variable **nombre** ;
4. maintenant qu'on a un pointeur sur **nombre**, on peut modifier directement la variable **nombre** en mémoire ! Il suffit d'utiliser ***pointeurSurNombre** pour désigner la variable **nombre** ! Pour l'exemple, on multiplie simplement la variable **nombre** par 3 ;
5. de retour dans la fonction main, notre **nombre** vaut maintenant 15 car la fonction **triplePointeur** a modifié directement la valeur de **nombre**.

Passage par adresse des paramètres d'une fonctions

En exécutant le programme du slide précédent, voici ce qu'il se passe dans l'ordre, en partant du début du main :

1. une variable **nombre** est créée dans le main. On lui affecte la valeur 5.
2. on appelle la fonction **triplePointeur**. On lui envoie en paramètre l'adresse de notre variable nombre ;
3. la fonction **triplePointeur** reçoit cette adresse dans **pointeurSurNombre**. À l'intérieur de la fonction **triplePointeur**, on a donc un pointeur **pointeurSurNombre** qui contient l'adresse de la variable **nombre** ;
4. maintenant qu'on a un pointeur sur **nombre**, on peut modifier directement la variable **nombre** en mémoire ! Il suffit d'utiliser ***pointeurSurNombre** pour désigner la variable **nombre** ! Pour l'exemple, on multiplie simplement la variable **nombre** par 3 ;
5. de retour dans la fonction main, notre **nombre** vaut maintenant 15 car la fonction **triplePointeur** a modifié directement la valeur de **nombre**.

Passage par adresse des paramètres d'une fonctions

En exécutant le programme du slide précédent, voici ce qu'il se passe dans l'ordre, en partant du début du main :

1. une variable **nombre** est créée dans le main. On lui affecte la valeur 5.
2. on appelle la fonction **triplePointeur**. On lui envoie en paramètre l'adresse de notre variable nombre ;
3. la fonction **triplePointeur** reçoit cette adresse dans **pointeurSurNombre**. À l'intérieur de la fonction **triplePointeur**, on a donc un pointeur **pointeurSurNombre** qui contient l'adresse de la variable **nombre** ;
4. maintenant qu'on a un pointeur sur **nombre**, on peut modifier directement la variable **nombre** en mémoire ! Il suffit d'utiliser ***pointeurSurNombre** pour désigner la variable **nombre** ! Pour l'exemple, on multiplie simplement la variable **nombre** par 3 ;
5. de retour dans la fonction main, notre **nombre** vaut maintenant 15 car la fonction **triplePointeur** a modifié directement la valeur de **nombre**.

Passage par adresse des paramètres d'une fonctions

En exécutant le programme du slide précédent, voici ce qu'il se passe dans l'ordre, en partant du début du main :

1. une variable **nombre** est créée dans le main. On lui affecte la valeur 5.
2. on appelle la fonction **triplePointeur**. On lui envoie en paramètre l'adresse de notre variable nombre ;
3. la fonction **triplePointeur** reçoit cette adresse dans **pointeurSurNombre**. À l'intérieur de la fonction **triplePointeur**, on a donc un pointeur **pointeurSurNombre** qui contient l'adresse de la variable **nombre** ;
4. maintenant qu'on a un pointeur sur **nombre**, on peut modifier directement la variable **nombre** en mémoire ! Il suffit d'utiliser ***pointeurSurNombre** pour désigner la variable **nombre** ! Pour l'exemple, on multiplie simplement la variable **nombre** par 3 ;
5. de retour dans la fonction main, notre **nombre** vaut maintenant 15 car la fonction **triplePointeur** a modifié directement la valeur de **nombre**.

Passage par adresse des paramètres d'une fonctions

En exécutant le programme du slide précédent, voici ce qu'il se passe dans l'ordre, en partant du début du main :

1. une variable **nombre** est créée dans le main. On lui affecte la valeur 5.
2. on appelle la fonction **triplePointeur**. On lui envoie en paramètre l'adresse de notre variable nombre ;
3. la fonction **triplePointeur** reçoit cette adresse dans **pointeurSurNombre**. À l'intérieur de la fonction **triplePointeur**, on a donc un pointeur **pointeurSurNombre** qui contient l'adresse de la variable **nombre** ;
4. maintenant qu'on a un pointeur sur **nombre**, on peut modifier directement la variable **nombre** en mémoire ! Il suffit d'utiliser ***pointeurSurNombre** pour désigner la variable **nombre** ! Pour l'exemple, on multiplie simplement la variable **nombre** par 3 ;
5. de retour dans la fonction main, notre **nombre** vaut maintenant 15 car la fonction **triplePointeur** a modifié directement la valeur de **nombre**.

Passage par adresse des paramètres d'une fonctions

En exécutant le programme du slide précédent, voici ce qu'il se passe dans l'ordre, en partant du début du main :

1. une variable **nombre** est créée dans le main. On lui affecte la valeur 5.
2. on appelle la fonction **triplePointeur**. On lui envoie en paramètre l'adresse de notre variable nombre ;
3. la fonction **triplePointeur** reçoit cette adresse dans **pointeurSurNombre**. À l'intérieur de la fonction **triplePointeur**, on a donc un pointeur **pointeurSurNombre** qui contient l'adresse de la variable **nombre** ;
4. maintenant qu'on a un pointeur sur **nombre**, on peut modifier directement la variable **nombre** en mémoire ! Il suffit d'utiliser ***pointeurSurNombre** pour désigner la variable **nombre** ! Pour l'exemple, on multiplie simplement la variable **nombre** par 3 ;
5. de retour dans la fonction main, notre **nombre** vaut maintenant 15 car la fonction **triplePointeur** a modifié directement la valeur de **nombre**.

Et si on revenait à notre exercice de départ ?

Solution

```
void decoupeMinutes(int, int* pointeurHeures, int*
    pointeurMinutes);
int main()
{
    int duree=0;
    int heures = 0, minutes =0 ;
    printf("Donnez le nombre de minutes : \n"); scanf("%d",
        &duree);
    // On envoie l'adresse de heures et minutes
    decoupeMinutes(duree, &heures, &minutes);
    // Cette fois, les valeurs ont ete modifiees !
    printf("%d heures et %d minutes", heures, minutes);
    return 0;
}
void decoupeMinutes(int fduree, int* pointeurHeures, int*
    pointeurMinutes)
{
    *pointeurHeures = fduree / 60;
    *pointeurMinutes = fduree % 60;
}
```

En résumé

- Chaque variable est stockée à une adresse précise en mémoire.
- Les pointeurs sont semblables aux variables. Au lieu de stocker un nombre ils stockent l'adresse à laquelle se trouve une variable en mémoire.
- Si on place un symbole **&** devant un nom de variable, on obtient son adresse au lieu de sa valeur (ex. : & age).
- Si on place un symbole * devant un nom de pointeur, on obtient la valeur de la variable stockée à l'adresse indiquée par le pointeur.
- Les pointeurs constituent une notion essentielle du langage C, mais néanmoins un peu complexe au début. Il faut prendre le temps de bien comprendre comment ils fonctionnent car beaucoup d'autres notions sont basées dessus.

En résumé

- Chaque variable est stockée à une adresse précise en mémoire.
- Les pointeurs sont semblables aux variables. Au lieu de stocker un nombre ils stockent l'adresse à laquelle se trouve une variable en mémoire.
- Si on place un symbole **&** devant un nom de variable, on obtient son adresse au lieu de sa valeur (ex. : & age).
- Si on place un symbole * devant un nom de pointeur, on obtient la valeur de la variable stockée à l'adresse indiquée par le pointeur.
- Les pointeurs constituent une notion essentielle du langage C, mais néanmoins un peu complexe au début. Il faut prendre le temps de bien comprendre comment ils fonctionnent car beaucoup d'autres notions sont basées dessus.

En résumé

- Chaque variable est stockée à une adresse précise en mémoire.
- Les pointeurs sont semblables aux variables. Au lieu de stocker un nombre ils stockent l'adresse à laquelle se trouve une variable en mémoire.
- Si on place un symbole **&** devant un nom de variable, on obtient son adresse au lieu de sa valeur (ex. : & age).
- Si on place un symbole * devant un nom de pointeur, on obtient la valeur de la variable stockée à l'adresse indiquée par le pointeur.
- Les pointeurs constituent une notion essentielle du langage C, mais néanmoins un peu complexe au début. Il faut prendre le temps de bien comprendre comment ils fonctionnent car beaucoup d'autres notions sont basées dessus.

En résumé

- Chaque variable est stockée à une adresse précise en mémoire.
- Les pointeurs sont semblables aux variables. Au lieu de stocker un nombre ils stockent l'adresse à laquelle se trouve une variable en mémoire.
- Si on place un symbole **&** devant un nom de variable, on obtient son adresse au lieu de sa valeur (ex. : & age).
- Si on place un symbole ***** devant un nom de pointeur, on obtient la valeur de la variable stockée à l'adresse indiquée par le pointeur.
- Les pointeurs constituent une notion essentielle du langage C, mais néanmoins un peu complexe au début. Il faut prendre le temps de bien comprendre comment ils fonctionnent car beaucoup d'autres notions sont basées dessus.

En résumé

- Chaque variable est stockée à une adresse précise en mémoire.
- Les pointeurs sont semblables aux variables. Au lieu de stocker un nombre ils stockent l'adresse à laquelle se trouve une variable en mémoire.
- Si on place un symbole **&** devant un nom de variable, on obtient son adresse au lieu de sa valeur (ex. : & age).
- Si on place un symbole ***** devant un nom de pointeur, on obtient la valeur de la variable stockée à l'adresse indiquée par le pointeur.
- Les pointeurs constituent une notion essentielle du langage C, mais néanmoins un peu complexe au début. Il faut prendre le temps de bien comprendre comment ils fonctionnent car beaucoup d'autres notions sont basées dessus.