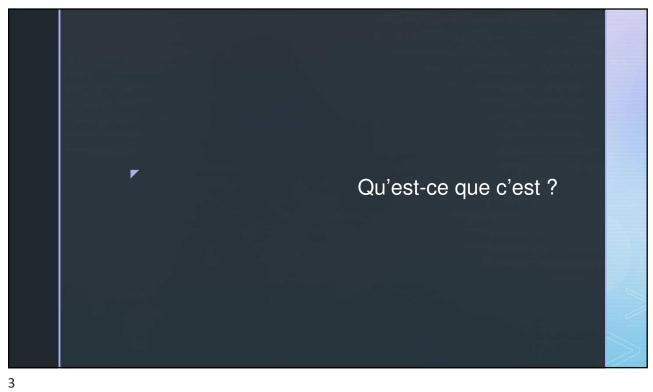


Notre premier défi! Les pointeurs

- Qu'est-ce que c'est?
- Comment connaître l'adresse mémoire d'une variable?
- Comment déclarer et utiliser un pointeur?
- Petits détails intéressants



	Rappe	elez-vous, la mémoire et les variables
Adresse	Espace d'encodage	13.13.5.55
0x1	null	
0x2		<ul> <li>Chaque espace contient 8 bits (1 octet)</li> </ul>
0x3	int	· · · · · · · · · · · · · · · · · · ·
0x4	int	<ul> <li>Une variable prend autant de cases que</li> </ul>
0x5		nécessaire
0x6	null	<ul> <li>Sa valeur sera consignée à l'intérieur</li> </ul>
0x7	null	<ul> <li>Le nombre de cases dépend du type</li> </ul>
0x8		
0x9	int	Supposons que les cases qui ne sont
0x10	IIIL	pas utilisées sont à <i>null</i>
0x11		
0x12	null	

•		Est-ce un problème ?
Adresse	Espace d'encodage	
0x1	null	The second secon
0x2		
0x3	int	<ul><li>Dans des cas simples, Non!</li></ul>
0x4		<ul> <li>Dans des cas plus complexes, Oui !</li> </ul>
0x5		
0x6	null	Parfois, il est plus simple de se rappeler
0x7	null	l'adresse de la case mémoire au lieu de
0x8		la variable au complet
0x9	int	Pourquoi d'après vous ?
0x10	(	
0x11		
0x12	null	

•		maginer un tableau de char
Adresse	Espace d'encodage	
0x1	null	The second secon
0x2		
0x3	char[4]	
0x4		
0x5		<ul> <li>Créons un tableau de 4 char</li> </ul>
0x6	null	<ul> <li>Imaginons que vous devions le copier</li> </ul>
0x7	null	C'est assez simple
0x8	null	0 63t 43362 3iiiipio
0x9	null	
0x10	null	
0x11	null	
0x12	null	

•		maginer un tableau de char
Adresse	Espace d'encodage	
0x1	null	
0x2		
0x3	char[4]	
0x4		
0x5		<ul> <li>Créons un tableau de 4 char</li> </ul>
0x6	null	<ul> <li>Imaginons que vous devions le copier</li> </ul>
0x7	null	C'est assez simple
0x8		- O Cot assez simple
0x9	- la - vF43	The second second
0x10	char[4]	
0x11		
0x12	null	
		- X -

	,		Imagine	er un tableau de char
Adresse	Espace d'encodage	Adresse	Espace d'encodage	
0x1	null	0x14		· · · · · · · · · · · · · · · · · · ·
0x2		0x15		
0x3		0x16		<ul> <li>Maintenant, créons un</li> </ul>
0x4		0x17		
0x5		0x18		tableau de 24 char
0x6		0x19	ahaw[04]	<ul><li>Imaginons que vous devions</li></ul>
0x7	char[24]	0x20	char[24	le copier
0x8	Char[24]	0x21		le copiei
0x9		0x22		<ul> <li>Ça devient déjà plus</li> </ul>
0x10		0x23		compliqué
0x11		0x24		- John Pilique
0x12		0x25		
0x13		0x26	null	
	. 14			

### Existe-t-il une solution? - Se rappeler où l'on a rangé la variable - Une variable de type pointeur conserve donc l'adresse où une variable est mémorisée - Le pointeur pointe vers l'emplacement mémoire d'une variable.

5



### Utilisez l'opérateur d'adresse & En C++, il est possible de connaître l'adresse mémoire de toutes les variables qui ont été déclarées. Peu importe le type Pour y parvenir, la syntaxe est simplement : &nom\_variable

• Regarder le code **adresse.cpp** pour avoir un exemple

11



### Déclarer un pointeur, c'est aussi simple que la déclaration de variable

• La syntaxe de déclaration d'un pointeur est aussi simple que :

```
type *nom_du_pointeur;
type *nom_du_pointeur1, *nom_du_pointeur2;
```

- Remarquez le symbole « \* » devant le nom du pointeur
  - C'est ce symbole qui permet la déclaration du pointeur
  - Le symbole « \* » doit être placé devant chaque variable
- Pour affecter une valeur à un pointeur, nous passons par l'opérateur d'adresse
  - Regarder le code **pointeur.cpp**

13

### Travailler sur la valeur pointée

- Grâce aux pointeurs, vous connaissez l'adresse d'une variable
- Si vous connaissez l'adresse d'une variable, vous pouvez modifier la valeur contenue à cette adresse
  - Si vous savez où les biscuits sont rangés, vous pouvez en prendre
- Pour y accéder, nous ajoutons le symbole « \* » devant la variable pointeur
  - Regarder le code pointeur.cpp

### Lire et écrire une valeur • Lire la valeur contenue à une adresse pointée Ex : float b = \*pointeur; • Écrire une valeur dans une adresse pointée Ex : \*pointeur = 6 + 4;

15

### L'autre avantage, la transmission de paramètre

- Dans le cours sur les fonctions, nous avons vu que les paramètres étaient transmis par valeur
  - Chaque variable était copiée
- Les pointeurs permettent un autre type de passage : le passage par référence
  - Les variables ne sont plus copiées
  - On transfère la référence vers ces valeurs
    - La référence n'est ni plus ni moins que les pointeurs

# Le passage par référence a une conséquence Comme il n'y a plus de copie, toute modification change directement la variable référée

17

### Le passage par référence passe par des paramètres de type pointeur

- Afin que vos fonctions acceptent un passage par référence, il faut programmer vos paramètres en conséquence
- Le passage par référence est disponible pour les types pointeurs seulement
- Dans l'exemple qui suit, la fonction PuissanceValeur aura un passage par valeur et la fonction PuissanceRefence aura un passage par référence
  - La différence est que tous les types attendus sont des pointeurs et non des valeurs

```
Passage par valeur

Passage par valeur

double PuissanceValeur(int a, int b) {
  int Resultat = 1;
  for (int i = 0; i < b; i++) {
    Resultat *= a;
  }
  return Resultat;
}

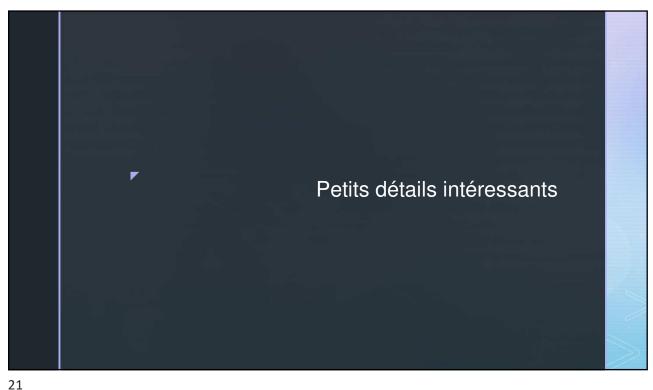
Exemple de passage

Passage par référence

double PuissanceValeur(int *a, int *b) {
  int Resultat = 1;
  for (int i = 0; i < *b; i++) {
    Resultat *= *a;
  }
  return Resultat;
}
```

### La fonction peut modifier son paramètre

- Grâce aux pointeurs, vous pouvez modifier directement une variable sans avoir de retour à effectuer
  - Regardez l'exemple dans fonctions.cpp
- Nous verrons d'autres applications avec les tableaux dans le prochain module



### Qu'est-ce que le type void\*? Nous avons que chaque type possède une version pointeur Le type pointeur void\* représente un pointeur général dont le type n'est pas spécifié • On ne peut pas utiliser ce type directement Il sert à définition de retour des fonctions • Pratique quand le type de pointeur n'est pas connu à l'avance Nous verrons ça dans le prochain module

### Renvoyez deux valeurs

- Normalement, une fonction ne peut pas renvoyer plus d'une valeur
- Les pointeurs permettent de passer outre cette limite
- Dans l'exemple renvoie.cpp, la fonction DecoupeMinute renvoie le nombre d'heures et elle modifie la variable minute afin de renvoyer deux informations

23

### L'indentation et les pointeurs

- L'opération d'accès à la valeur (\*) a la même priorité que les opérateurs unitaires (!, ++, --)
- Leur calcul est réalisé de droite à gauche
- Imaginons les variables int r = 5; int \*p = &r;
- Il est possible d'incrémenter r sans problème avec la notation préfix

++\*p;

• On accède à la valeur contenue à l'Adresse de p et **après** on incrémente (de droite à gauche)

### L'inverse est faux Maintenant, imaginons l'opération suivante \*p++; L'incrémentation est réalisée avant sur le pointeur Après on accède à la case mémoire Note : L'incrémentation d'un pointeur correspond à déplacer l'adresse d'un nombre d'octets équivalent à son type Pour un int, ceci équivaut à 4 octets La bonne notation est donc : (\*p)++ Les parenthèses ont la plus grande priorité

25

# Le pointeur NULL Il existe un pointeur spécial Contenu dans iostream Ce pointeur ne pointe nulle part Il s'agit du pointeur NULL int \*p = NULL; if (p == NULL) cout << "p ne pointe nulle part" << endl;</li>

