Les tableaux dynamiques, une question de pointeurs

420-C21-IN Programmation II

Godefroy Borduas – Automne 2021

Module 5

1

Une histoire d'opérateur

- Le lien entre tableau et pointeur
- Accéder aux valeurs d'un tableau à l'aide de pointeur
- Comparaison de pointeur
- Tableau à deux dimensions
- Allocation dynamique de la mémoire
- Petite précaution

2

Le lien entre tableau et pointeur

3

Un tableau n'est qu'un emplacement mémoire

- Rappelez-vous, un tableau est un espace mémoire qui rassemble plusieurs blocs du même type
- Exemple, un tableau de 10 entier sera un espace de 10 blocs d'entier (4 octets)
- Si un espace mémoire possède une adresse, alors chaque bloc mémoire (qui sont des espaces individuels) aura une adresse
- Nous pouvons donc résumer un tableau à une adresse
  - Celle de la première case, le point de départ du tableau

4

Allure d'un tableau

- Imaginez un tableau de char
- Donnons-lui 13 cases
- En mémoire, le tableau ressemble à ça :
- En gros : tableau[0] et tableau ont la même adresse

Tableau

Adresse	Valeur
0x..1	null
0x..2	Tableau[0]
0x..3	Tableau[1]
0x..4	Tableau[2]
0x..5	Tableau[3]
0x..6	Tableau[4]
0x..7	Tableau[5]
0x..8	Tableau[6]
0x..9	Tableau[7]
0x..A	Tableau[8]
0x..B	Tableau[9]
0x..C	Tableau[10]
0x..D	Tableau[11]
0x..E	Tableau[12]
0x..F	null
0x..10	null

5

Comment déclarer un pointeur pour un tableau

- Nous pouvons passer par la première case

```
float A[10];
float *p;
P = &A[0];
```

  - Avec cette méthode, l'opérateur d'adresse est obligatoire, car la case 0 du tableau est une variable et non une référence à l'adresse mémoire
- Ou nous pouvons passer par le tableau lui-même (qui n'est ni plus ni moins qu'un pointeur)

```
float A[10];
float *p;
P = A;
```

6

Accéder aux valeurs d'un tableau à l'aide de pointeur

7

Comment fonctionne l'arithmétique d'un pointeur ?

- Les pointeurs contiennent une adresse mémoire
- Incrémenter un pointeur revient à incrémenter l'adresse mémoire
- Prenons un cas simple

```
Pointeur => 0x00010
Pointeur + 1 => 0x0011
Pointeur + 3 => 0x0013
Pointeur + 10 => 0x001A
```

- Les adresses sont en hexadécimale
- Remarquez, l'adresse augmente d'un octet à chaque fois
  - Or, certains types ont une taille plus grande qu'un octet

8

Est-ce que l'ajout d'une taille mémoire sera toujours d'un octet ?

- Non, loin de là
- Le compilateur sait, d'après le type du pointeur, de quelle taille il doit augmenter
- Pour un pointeur entier, il augmente de 4 octets

```
PointeurEntier => 0x00010
PointeurEntier + 1 => 0x00014
```

- Pour un pointeur double, il augmente de 8 octets

```
PointeurEntier => 0x00010
PointeurEntier + 1 => 0x00018
```

9

En résumé, les opérations sur le pointeur P deviennent…

- `P + 1` : Augmente l'adresse mémoire d'une taille de variable
- `P + n` : Augmente l'adresse mémoire de n fois la taille de variable
- `P++` : Augmente l'adresse mémoire d'une taille de variable et l'affecte au pointeur
- `P += n` : Augmente l'adresse mémoire d'une taille de variable et l'affecte au pointeur
- `P - 1` : Diminue l'adresse mémoire d'une taille de variable
- `P - n` : Diminue l'adresse mémoire de n fois la taille de variable
- `P--` : Diminue l'adresse mémoire d'une taille de variable et l'affecte au pointeur
- `P += n` : Diminue l'adresse mémoire d'une taille de variable et l'affecte au pointeur

10

Disons dans le cas d'un tableau

- SI

  `float A[10]; float* P = A;`

- Alors, nous aurons les équivalents suivants :

  `*P ⇔ A[0]`

  `*(P + 0) ⇔ A[0]`

  `*(P + 4) ⇔ A[4]`

- Ces opérateurs (+, -, ++, --, +=, -=) sont définis seulement **à l'intérieur d'un tableau**, car on en peut pas présumer que2 variables de même type sont stockées de façon continue en mémoire.
- Note : le tableau **A** est une constante, le pointeur **P** est une variable
  - On peut donc modifier **P**, mais ce n'est pas possible pour **A** (il faut passer par les indices)

11

Distinction importante

Adresse de la case i du tableau A (*P = A)

```
P + i
```

Valeur de la case i du tableau A (*P = A)

```
*(P + i)
```

12

Comparaison de pointeur

13

Comparer deux pointeurs

- Regarder le code **[inscrirelenomici].cpp**
- Comparer deux pointeurs d'un même tableau ⇔ comparer les indices d'un même tableau
- Comparer deux pointeurs de deux tableaux ⇔ comparer les positions **relatives** des tableaux dans la mémoire

14

Tableau à deux dimensions

15

Rappelons-nous ce qu'est une matrice

Code C++ de la matrice

```
int M[4][3] = {   {1, 2, 3},
                  {4, 5, 6},
                  {7, 8, 9}};
```

Exemple en mémoire

Adresse	Valeur
0x..1	null
0x..2	M[0]
0x..E	M[1]
0x..1A	M[2]
0x..26	M[3]
0x..32	M[4]
0x..33	null

16

Chaque ligne est une case, mais qu'est-ce qui arrive à chaque colonne de la ligne ?

- Chaque cellule est une case mémoire
- En somme, la case M[0] pointe vers un tableau

Adresse	Valeur
0x..1	null
0x..2	M[0]
0x..E	M[1]
0x..1A	M[2]
0x..26	M[3]
0x..32	M[4]
0x..33	null

Adresse	Valeur
0x..1	null
0x..2	M[0][0]
0x..6	M[0][1]
0x..A	M[0][2]
0x..E	M[1]

17

Quel est l'impact pour les pointeurs

- Il faut pointer sur la première case du tableau

```
int *P = M[0];
```

- Pour le parcourir, il suffit de passer les indices du pointeur

```
Pour i = 0; i < NbLigne + NbColonne; i++
    Afficher *(P + i)
```

- On peut aussi passer par la conversion forcée

```
int *P = (int *)M;
```

18

Allocation dynamique de la mémoire

19

Depuis le début, on utilise l'**allocation statique**

- Allouer la mémoire ⇔ Attribuer une case mémoire à une variable
- Jusqu'à maintenant, c'est le compilateur qui s'est occupé de l'allocation de mémoire
- C'est possible, car le compilateur connaît la taille de mémoire nécessaire pour les types simples et les tableaux
- C'est ce qu'on appelle l'**allocation statique**
  - Car l'attribution ne dépend pas de la situation
- Le même principe s'applique aux chaînes constantes
  - Le nombre de caractères a tout une taille identique

20

Le but de l'**allocation dynamique**

- Dans certain cas, la taille de mémoire nécessaire n'est pas connue à l'avance
- Prenons l'exemple d'un registre d'ambassadeur dans une station spatiale en zone neutre
  - Imaginez que l'on place ce registre dans un tableau
  - On ne connaît pas le nombre d'ambassadeur avant la compilation
  - La taille du tableau sera donc ajustée au moment de sa création **durant l'exécution**
- On parle alors d'**allocation dynamique**
- C'est-à-dire que l'attribution de la mémoire va dépendre de la situation

21

Allocation dynamique en C++

- L'allocation dynamique crée un **pointeur**
- En C++, l'allocation passe par le mot clé **new**
- La syntaxe générale pour un type **simple** :

```
<type> *<nom> = new <type>;
```

- La syntaxe générale pour un type **tableau** :

```
<type> *<nom> = new <type>[<taille>];
```

- Pour le reste, on applique les notions sur les pointeurs
- **ATTENTION !** Il faut libérer la mémoire.

22

Libérer la mémoire en C++

- Il faut libérer la mémoire avant la fin du programme
  - Lors de l'attribution, l'OS donne une partie de la mémoire au programme
  - Il faut redonner cette mémoire à l'OS
  - Si ce n'est pas fait, la mémoire sera utilisée jusqu'à ce que l'ordinateur se ferme
- Le mot clé **delete** permet de libérer la mémoire
- La syntaxe générale pour un type simple :

```
delete <identifiant_du_pointeur>;
```

- La syntaxe générale pour un tableau :

```
delete[] <identifiant_du_pointeur_tableau>;
```

23

Allocation dynamique en C pur

- L'allocation dynamique crée un **pointeur**
- En C pur, l'allocation passe par la méthode **malloc**
- La syntaxe générale :

```
<type> *<nom> = malloc(<taille_souhaitée>);
```

- Pour connaître la taille d'un type, on utilise la méthode **sizeof**

```
int TailleEnOctet = sizeof(<type>);
```

- Exemple pour un type simple :

```
float *Simple = malloc(sizeof(float));
```

- Exemple pour un tableau de quatre cases :

```
float *Tableau = malloc(sizeof(float) * 4);
```

  - Un tableau a la même taille que le nombre de case multiplier par la taille du type
  - Ceci fonctionne aussi avec une variable

Contenu dans `<stdlib.h>`

24

Libérer la mémoire en C pur

- Il faut libérer la mémoire avant la fin du programme
  - Lors de l'attribution, l'OS donne une partie de la mémoire au programme
  - Il faut redonner cette mémoire à l'OS
  - Si ce n'est pas fait, la mémoire sera utilisée jusqu'à ce que l'ordinateur se ferme
- La fonction **free** permet de libérer la mémoire
- La syntaxe générale :

```
free(<identifiant_du_pointeur>);
```

25

Petite précaution

26

Vérifiez toujours si le pointeur pointe sur une case

- S'il y a un problème, le pointeur sera vide (NULL)
- Avant d'utiliser un pointeur, il faut le vérifier

```
if (Pointeur != NULL)
    cout << "Le pointeur a été initialisé.";
```

27

Exercices

28

Exercice sur les pointeurs

- Écrivez un programme qui utilise la notion de pointeur pour lire deux entiers et calculer leur somme.
- Écrivez un programme qui utilise la notion de pointeur pour calculer la moyenne d'un ensemble de valeur d'un tableau de nombre flottant. Laissez les personnes utilisatrices rentrez les valeurs du tableau.
- Écrivez un programme qui réserve l'espace mémoire à un tableau de caractères sous forme d'un triangle droit, le remplit par des étoiles (*) puis l'affiche.

```
*
**
****
******
********
```

29