Algorithmique et Structures de données

Thomas Lavergne & Florent Hivert

Mél : thomas.lavergne@lisn.upsaclay.fr

Passage de paramètres...

Lors de l'appel d'une fonction ou procédure, les étapes suivantes sont exécutées :

Retenir

1. *Empilement dans la mémoire de son tableau d'activation (**dont les emplacements nécessaires pour ses paramètres**) ;*
2. ***Initialisation des valeurs des paramètres** («passage des paramètres»), et donc modification de la mémoire ;*
3. *Exécution du corps de la fonction, dans l'environnement : la mémoire est ainsi modifiée ;*
4. *Pour les fonctions, retour de la valeur de la fonction ;*
5. *Dépilement du bloc d'activation et retour à l'environnement qui était en place avant l'appel. Seul l'environnement est restauré, les effets de la procédure sur la mémoire persistent.*

Pourquoi passer un paramètre ?

Les paramètres servent à communiquer avec la fonction.

Retenir

Il peut y avoir trois raison de communiquer :

1. *on veut faire passer **une donnée** à la fonction*
2. *on veut récupérer **un résultat** de la fonction*
3. *on veut que la fonction **modifie une variable** qui a déjà une valeur.*

Exemple de passage d'une **donnée**

Fonction calculant la somme de deux entiers :

```
int somme(int a, int b);
```

Les paramêtres `a` et `b` sont des **données** de la fonction.

Exemple de passage d'un résultat

On a souvent besoin d'une fonction qui retourne **deux** résultats. Pour le moment, avec l'instruction `return`, on ne peut en retourner qu'un seul.

Fonction qui retourne si un entier est une puissance de deux et qui renvoie l'exposant de la plus petite puissance supérieure ou égale à l'entier :

- pour 4, on doit répondre : vrai, 2
- pour 9, on doit répondre : faux, 4

Exemple de passage d'un **résultat**

Fonction qui retourne si un entier est une puissance de deux et qui renvoie l'exposant de la plus petite puissance supérieure ou égale à l'entier :

```
bool estPuissanceDe2(int n, int &exp);
```

- Le paramètre `n` est une **donnée**
- Le paramètre `exp` est un **résultat**

Remarque

- Noter le «&» devant le paramètre `exp`.
- Lors de l'appel, la variable `exp` n'est **pas initialisée**, c'est le rôle de la fonction `estPuissanceDe2` de le faire.

Exemple de passage d'un **résultat**

```
 1  /** Teste si un nombre est une puissance de 2 et calcule l'exposant.
 2   * @param[in]  n    un nombre entier positif
 3   * @param[out] exp  l'exposant de la plus petite
 4   *                  puissance de 2 supérieure ou égale à n
 5   * @return un boolean
 6   **/
 7  bool estPuissance2(int n, int &exp) {
 8      exp = 0;
 9      int puiss = 1;
10      while (puiss < n) {
11          exp++;
12          puiss *= 2;
13      }
14      return puiss == n;
15  }
16
17  int main() {
18      int a, e;
19      bool b;
20      cin >> a;
21      b = estPuissance2(a, e);
22      // Le booléen b est affiché avec 0 pour false et 1 pour true
23      cout << b << " " << e << endl;
24  }
```

Exemple de passage de **donnée/résultat**

C'est le cas où l'on veut que la fonction modifie une variable qui contient déjà une valeur :

- fonction `incrémente` qui fait la même chose que ++ :

  ```
  void incrémente(int &n)
  ```

- fonction `échange` qui échange le contenu de deux variables :

  ```
  void échange(int &n, int &m)
  ```

Remarque

- Le C++ ne fait **pas la différence** entre les modes de passage résultat et donnée/résultat.
- Dans le mode donnée/résultat, la variable doit être **initialisée avant l'appel**.

Exemple de passage de **donnée/résultat**

```
 1  /** Ajoute 2 à un entier
 2  * @param[in/out] a une variable entière
 3  **/
 4  void ajoute2(int &a) {
 5      a = a + 2;
 6  }
 7
 8
 9  int main() {
10      int n;
11      cin >> n;
12      cout << "Avant :  " << n << endl;
13      ajoute2(n);
14      cout << "Apres :  " << n << endl;
15  }
```

Exemple de passage de **donnée/résultat**

```
 1  /** Échange les contenus de deux variables
 2   * @param[in/out] a b deux variables entières.
 3   **/
 4  void echange(int &a, int &b) {
 5      int t;
 6      t = a;
 7      a = b;
 8      b = t;
 9  }
10
11  int main() {
12      int x, y;
13      cin >> x >> y;
14      cout << "Avant :  " << x << " " << y << endl;
15      echange(x, y);
16      cout << "Apres :  " << x << " " << y << endl;
17  }
```

Passage de paramètres...

Deux modes principaux de passage de paramètres sont utilisés :

Retenir

- *le **passage par valeur**, utilisé pour passer une donnée. En C++, c'est le mode de passage par défaut.*
- *le passage par référence (ou par adresse) : correspond aux paramètres résultats ou données-résultats.*
  En C++, ce mode de passage est obtenu par l'utilisation de références (symbole «&»).

Passage de paramètres...

Deux modes principaux de passage de paramètres sont utilisés :

Retenir

- *le **passage par valeur**, utilisé pour passer une donnée. En C++, c'est le mode de passage par défaut.*
- *le **passage par référence** (ou par adresse) : correspond aux paramètres résultats ou données-résultats.*
  En C++, ce mode de passage est obtenu par l'utilisation de références (symbole «&»).

Pour reporter les modifications, on fait un passage par référence

On en a besoin pour un paramètre en mode résultat ou donnée/résultat.

Notion de référence

Le C++ permet de manipuler un type particulier, dit «référence». **Une référence est un identificateur synonyme d'un autre identificateur**. Elle permet de manipuler une variable sous un autre nom que celui sous lequel cette variable a été déclarée.

Syntaxe

On déclare une référence par

$$type\ \&nom = variable\,;$$

Par exemple :

```
1       int i=1;
2       int  &ri = i; // référence sur la variable i
3       ri = ri * 2; // double la valeur de ri et de i !!
```

Référence

Dans l'exemple :

```
1       int i=1;
2       int  &ri = i; // référence sur la variable i
3       ri = ri * 2; // double la valeur de ri et de i !!
```

Les deux identificateurs `i` et `ri` **représentent alors la même variable** (le même emplacement mémoire) qu'ils peuvent tous les deux **accéder et modifier.**

Référence

Retenir

*En pratique, une **variable de type référence contient l'adresse de la variable référencée**.*

Nous utiliserons les conventions de dessin suivantes :

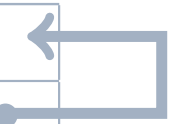
- *Dans l'environnement, les variables de type référence seront précédées du symbole &.*
- *On dessinera une flèche entre la référence et la variable référencée.*

```
1       int i=1;
2       int  &ri = i; // référence sur la variable i
3       ri = ri * 2; // double la valeur de ri et de i !!
```

i	4	1
&ri	3	4

sera représenté par

i	4	1
&ri	3	



Référence

Retenir

*En pratique, une **variable de type référence contient l'adresse de la variable référencée**.*
Nous utiliserons les conventions de dessin suivantes :

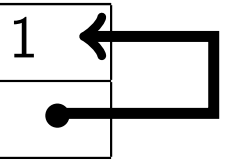
- *Dans l'environnement, **les variables de type référence seront précédées du symbole** `&`.*
- *On dessinera une flèche entre la référence et la variable référencée.*

```
1       int i=1;
2       int  &ri = i; // référence sur la variable i
3       ri = ri * 2; // double la valeur de ri et de i !!
```

i	4	1
&ri	3	**4**

sera représenté par

i	4	1
&ri	3	●



Passage par référence

Comme dans le passage par valeur, des emplacements sont prévus dans les tableaux d'activation pour être alloués aux paramètres passés par référence.

Retenir

Passage par référence *Mais, au contraire du passage par valeur où la valeur du paramètre effectif est recopiée dans le nouvel emplacement,* ***on transfère ici la référence*** *(l'adresse) du paramètre effectif. Le paramètre formel et le paramètre effectif correspondent alors* ***au même emplacement.***

Passage par référence

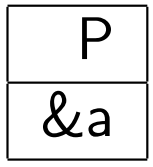
Comme dans le passage par valeur, des emplacements sont prévus dans les tableaux d'activation pour être alloués aux paramètres passés par référence.

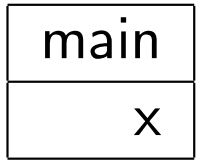
Retenir

Passage par référence *Mais, au contraire du passage par valeur où la valeur du paramètre effectif est recopiée dans le nouvel emplacement,* ***on transfère ici la référence*** *(l'adresse) du paramètre effectif. Le paramètre formel et le paramètre effectif correspondent alors* ***au même emplacement.***

Exemple de Passage par référence

```
1  void P(int &a) {
2    cout << "a =  " << a << endl;
3    a = 0;
4  }
5  void main () {
6    int x;
7    /* P(10) serait une erreur */
8    x = 5;
9    P(x);
10   cout << "x = "  << x << endl;
11 }
```





Remarquez qu'ici, l'appel à `P(10)` a été supprimé car 10 est une constante (et pas un identificateur de variable). Donc on ne peut pas lui associer une référence....

Exemple de passage par référence

Entrée dans la fonction `main`

```
1  void P(int &a) {
2    cout << "a =  " << a << endl;
3    a = 0;
4  }
5  void main () {
6    int x;
7    /* P(10) serait une erreur */
8    x = 5;
9    P(x);
10   cout << "x = "  << x << endl;
11 }
```

Pile :

	7	?
	6	?
	5	?
	4	?
	3	?
	2	?
main	1	ADM
x	0	?

Exemple de passage par référence

`main` ligne 8 :

```
 1  void P(int &a) {
 2    cout << "a =  " << a << endl;
 3    a = 0;
 4  }
 5  void main () {
 6    int x;
 7    /* P(10) serait une erreur */
 8    x = 5;
 9    P(x);
10    cout << "x = "  << x << endl;
11  }
```

Pile :

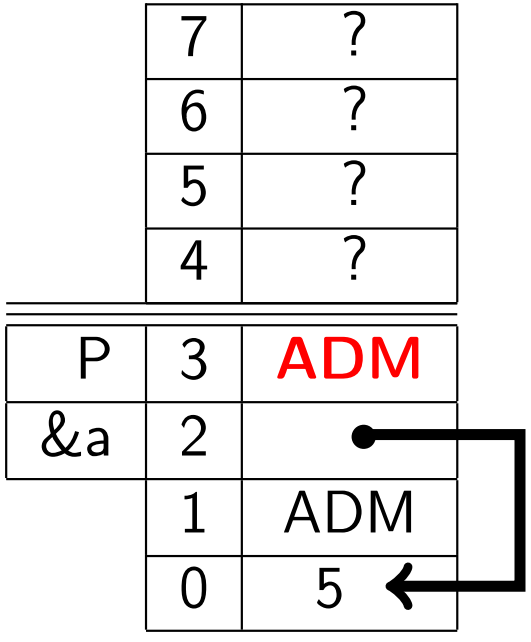
	7	?
	6	?
	5	?
	4	?
	3	?
	2	?
main	1	ADM
x	0	**5**

Exemple de passage par référence

`main` ligne 9 : appel de `P(x)`, c'est-à-dire `P(&0)`

```
 1  void P(int &a) {
 2    cout << "a =  " << a << endl;
 3    a = 0;
 4  }
 5  void main () {
 6    int x;
 7    /* P(10) serait une erreur */
 8    x = 5;
 9    P(x);
10    cout << "x = "  << x << endl;
11  }
```

Pile :

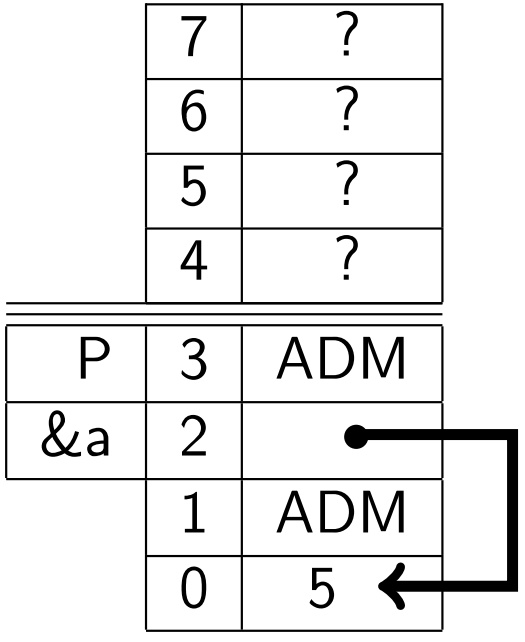


Exemple de passage par référence

Ligne 2 ; affichage de «a = 5»

```
1  void P(int &a) {
2    cout << "a =  " << a << endl;
3    a = 0;
4  }
5  void main () {
6    int x;
7    /* P(10) serait une erreur */
8    x = 5;
9    P(x);
10   cout << "x = "  << x << endl;
11 }
```

Pile :

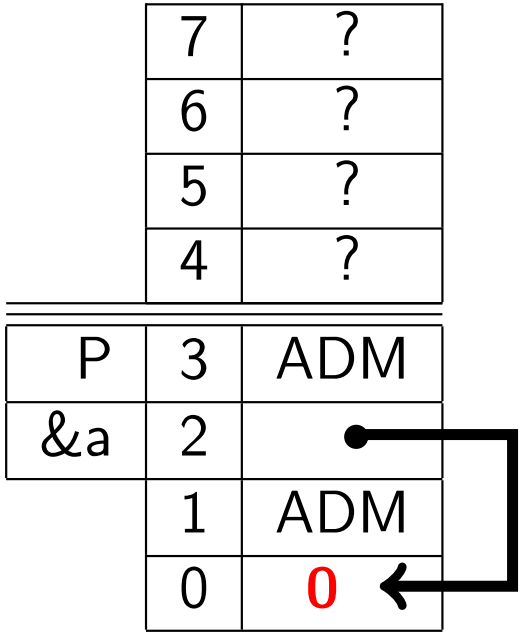


Exemple de passage par référence

Ligne 3

```
1  void P(int &a) {
2    cout << "a =  " << a << endl;
3    a = 0;
4  }
5  void main () {
6    int x;
7    /* P(10) serait une erreur */
8    x = 5;
9    P(x);
10   cout << "x = "  << x << endl;
11 }
```

Pile :

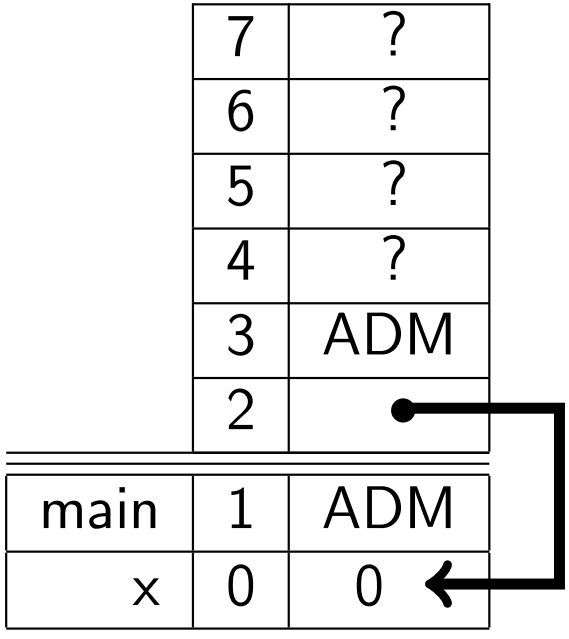


Exemple de passage par référence

Retour au `main` ligne 10 : affichage de «`x = 0`»

```
1  void P(int &a) {
2    cout << "a =  " << a << endl;
3    a = 0;
4  }
5  void main () {
6    int x;
7    /* P(10) serait une erreur */
8    x = 5;
9    P(x);
10   cout << "x = "  << x << endl;
11 }
```

Pile :



Bilan : passage par valeur / référence

Retenir

	Valeur	*Référence*
Param. réel	*valeur ou variable* *copie*	*variable* *adresse*
modifications	*non reportées*	*reportées*
utilisation	*donnée*	*don/rés ou rés* $*$

Référence vers un objet constant

Dans le cas d'un passage par valeur, la copie peut être coûteuse si l'objet passé est complexe (structure, vecteur...)

Compléments

Dans certains cas, on fait un **passage par référence pour éviter la copie**. Le C++ permet alors d'**interdire la modification** en déclarant l'objet passé par référence constant grâce au mot clé `const`.

```
float moyenne(vector<float> const &notes)
```

Référence vers un objet constant

Dans le cas d'un passage par valeur, la copie peut être coûteuse si l'objet passé est complexe (structure, vecteur...)

Compléments

Dans certains cas, on fait un **passage par référence pour éviter la copie**. Le C++ permet alors d'**interdire la modification** en déclarant l'objet passé par référence constant grâce au mot clé `const`.

```
float moyenne(vector<float> const &notes)
```

Adresse d'une référence

Retenir

En C++, l'adresse d'une référence est l'adresse de la variable référencée.

Le programme :

```
1  void main () {
2    int x = 5;
3    int &y = x;
4    cout << &x << " " << &y << endl;
5  }
```

affiche deux fois la même adresse.

Passage d'une référence

Retenir

En conséquence, on peut passer une ***référence comme paramètre réel d'un passage par référence****.*

```
1  void p2(int &n) {
2      cout << "p2 " << &n << endl;
3  }
4  void p1(int &m) {
5      cout << "p1 " << &m << endl;
6      p2(m);
7  }
8  int main() {
9      int a = 2;
10     cout << "main " << &a << endl;
11     p1(a);
12 }
```

Dans P2 :