420-V31-SF – Programmation de Jeux Vidéo III

Utilisation de const

fparadis@cegep-ste-foy.qc.ca

Quelques utilisations de const

3 utilisations de **const**…

1. pour un paramètre
2. pour une méthode
3. pour un retour de valeur en référence constante
4. (Pour un retour par valeur constant… possible mais ne sert strictement à rien).

①

```
float CalculerMoyenne(const int notes[])
{

}
```

③ ②

```
const bitmapImg &Joueur::GetImg() const
{
        return imageJoueur;
}
```

Utilisation de const

1- Pour un paramètre

Lorsqu'un paramètre n'a pas à être modifié par une fonction, il est fortement recommandé de le spécifier à l'aide du mot clé ***const***

Au lieu de :

```
float CalculerMoyenne(unsigned int scores[])
{
   /* le tableau scores peut être modifié */
   /* une instruction comme scores[0] = 42 est possible */
}
```

On devrait avoir :

```
float CalculerMoyenne(const unsigned int scores[])
{
   /* Impossible de modifier le tableau scores*/
    /* une instruction comme scores[0] = 42 est impossible */
}
```

Utilisation de const

2- Pour une méthode

Une **méthode** déclarée **const**, indique que cette méthode ne doit pas modifier l'état de l'objet.

```
class Joueur
{
  public:
    Joueur();
    ~Joueur();

    void SetNom(const string nom);
    string GetNom() const;

  private:
    string nom;
    int pointage;
};
```

```
void Joueur::setNom(const string nom)
{
   nom += " ";  // Impossible
   this->nom = nom;
}
```

```
string Joueur::GetNom() const
{
   this->nom += " ";  // Impossible
   return nom;
}
```

Utilisation de const

2- Pour une méthode (suite)

Une méthode **const** ne peut pas appeler une autre méthode qui **n'est pas const**.

```
void SetValeur(int nouvelleValeur)
{
    …
}

void FaireQuelqueChose(int nouvelleValeur) const
{
    SetValeur(nouvelleValeur); // INTERDIT car SetValeur n'est pas const

    this->valeur = nouvelleValeur; //INTERDIT car modifie l'état de l'objet

}
```

En résumé, chaque fois qu'une méthode ne modifie aucune donnée, il faudrait que **la méthode** soit **const**.

③

Utilisation de const

Donc systématiquement

- Vos get devraient être des méthodes constantes
- Vos set devraient avoir des paramètres constants

Permet d'améliorer la qualité du code de façon très importante.

Est une norme courante qui, lorsqu'utilisée, démontre le sérieux et la qualité de l'équipe de développement qui développe le code.

La librairie standard du C++ adhère à cette norme.

Le compilateur fera la vérification que les attributs de l'objet ne seront pas modifiés dans cette méthode.

③

Utilisation de const

Mais même encore

- Vous pourriez prendre l'habitude de mettre vos paramètres constants et de l'enlever si vous avez vraiment besoin de le modifier (normalement ça devrait être exceptionnel ou alors si le paramètre est passé par référence)

- Si une méthode est une simple méthode "de lecture" , alors elle devrait être const

③

Utilisation de const

3 - Pour un retour de valeur (attention plus complexe – suivez bien)

- Mettre const avant le type de retour fait que ce que retourne une fonction sera une constante.

- Peu utilisé sauf pour les retours de référence.
  - Pourquoi, parce que dans les faits, ça ne sert à rien
  - const int x =10;
  - int y = x; //que x soit const ne change strictement rien.

③

Utilisation de const

3 - Pour un retour de valeur (attention plus complexe – suivez bien)

- Si vous avez une fonction qui retourne une référence sur un objet crée dans la fonction, la référence ne référera plus à rien, puisque l'objet original est détruit à la fin de la fonction.

- Mais si vous retournez une référence constante, celui-ci sera conservé en mémoire.

- Par contre de l'autre côté, il faudra l'assigner également à une référence constante.

- Reste que c'est une pratique plutôt obsolète; à présent c'est plutôt une mauvais pratique de retourner des références sur des objets créés dans la méthode.

Utilisation de const

Pour un retour de valeur

Même principe que pour un paramètre en référence constante. Une **valeur de retour** en **référence constante** n'implique aucune copie de l'objet.

```
class Joueur
{
  public:
    Joueur();
    ~Joueur();
    void SetNom(const string _nom);
    string GetNom() const;
    const bitmapImg &GetImg() const;

  private:
    string nom;
    int pointage;
    BitmapImg imageJoueur;
};
```

```
const BitmapImg &Joueur::GetImg() const
{
        Joueur imageJoueur;
        return imageJoueur;
}
```

Aucune copie n'est retournée.
C'est la référence qui est retournée.

Attention, l'appel peut créer une copie. Exemples d'appels de la méthode GetImg()…

```
BitmapImg image = unJoueur.GetImg(); // Pas de & : Attention, il y a copie de l'objet. Lourd
                                     //mais pas grave si l'objet a été crée au milieu de la méthode

BitmapImg &image = unJoueur.GetImg(); // Ne compile pas, const vs non-const

const BitmapImg &image = unJoueur.GetImg(); // Mieux (on retourne la référence et c'est bien
une référence const qui la reçoit)
```

420-V31-SF – Programmation de Jeux Vidéo III

Passage de paramètres: valeur vs référence

fparadis@cegep-ste-foy.qc.ca

Passage de paramètres (valeur)

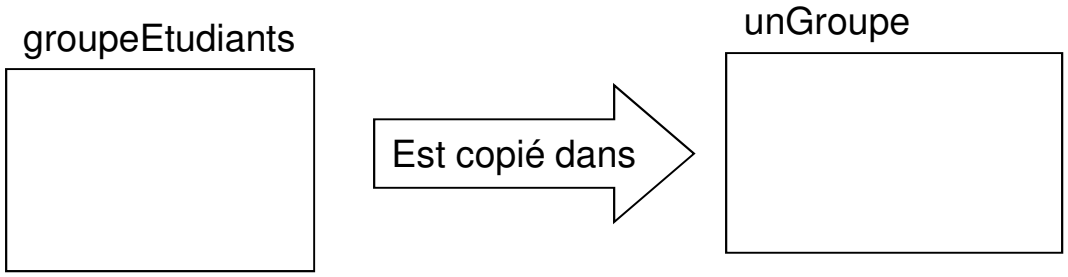
Par valeur:

Une copie indépendante de l'original est créée.

```
float CalculerMoyenne(Groupe unGroupe)
{
        /* ... */
}

resultat = CalculerMoyenne(groupeEtudiants);
```

Une copie de l'objet groupeEtudiants est faite dans l'objet unGroupe.



Remarque

Étant donné la copie de l'objet, le **mode par valeur** ne devrait idéalement être utilisé que pour les « petits » types de données (int, float, double, etc).

string n'est pas considéré comme un petit type.

Passage de paramètres (référence)

Par référence:

L'original est passé et peut être modifié.

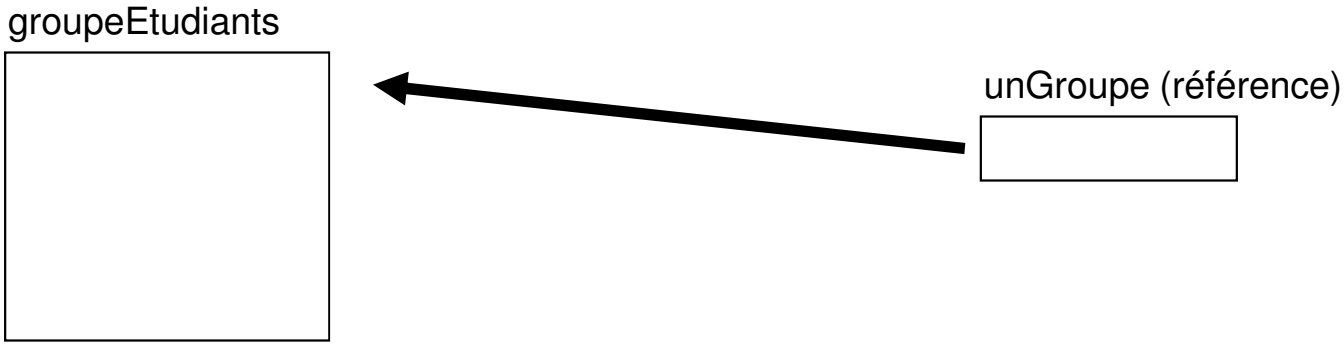
```
float CalculerMoyenne(Groupe &unGroupe)
{

    unGroupe.nbNote = 20;
    /* ... */
}


resultat = CalculerMoyenne(groupeEtudiants);
```

Cette modification a pour effet de modifier la propriété nbNote de **groupeEtudiants**.

Dans l'exemple ci-dessus, **unGroupe** devient une référence sur **groupeEtudiants**.
Toute modification sur **unGroupe** modifie **groupeEtudiants.**



Passage de paramètres (référence)

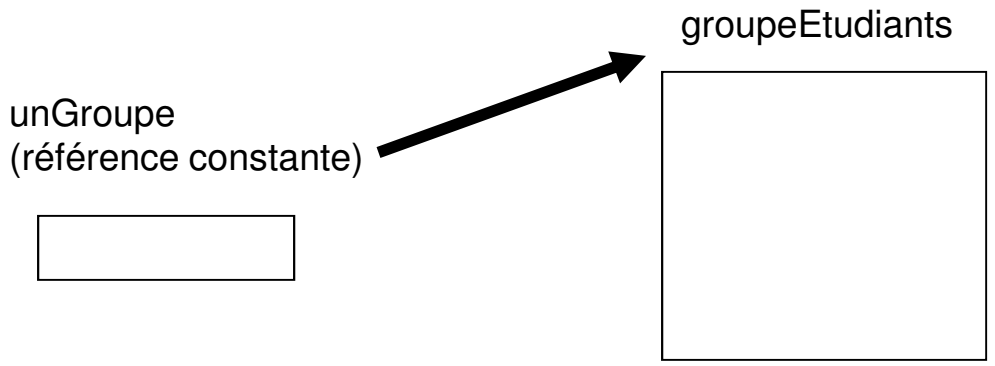
Par référence constante:

L'original est passé, mais ne peut pas être modifié.

```
float CalculerMoyenne(const Groupe &unGroupe)
{
          /* Il est impossible de faire:  */
           unGroupe.nbNotes = ...
}
/* ... */

resultat = CalculerMoyenne(groupeEtudiants);
```

Dans l'exemple ci-dessus, **unGroupe** devient une référence constante sur **groupeEtudiants**.
Il est alors impossible de modifier **unGroupe**.



Remarques sur le passage par référence (constante ou non)

Le mode par **référence (constante ou non)** devrait toujours être utilisé autant que possible pour les classes ou les structures. Il évite de faire des copies inutiles lorsque le type de données occupe beaucoup de mémoire.

Le passage **par référence** permet à une fonction de **retourner** plusieurs résultats.
Mais attention, à utiliser avec parcimonie, ou avec beaucoup de commentaires (lisibilité)

```
bool CalculerPosition(float &x,                    // En entrée / sortie
                      float &y,                    // En entrée / sortie
                      const float vitesse);        // En entrée

//Le retour booléen pourrait déterminer si c'est un mouvement légal
//Dans les faits on se retrouve avec trois sorties de fonction:
//dangeureux.
```

Attention ! **Un tableau est toujours passé par référence**. Il ne faut pas mettre &.

```
void uneFonction(Point p1[]);       // par référence
void uneFonction(const Point p1[]); // par référence constante
```

Passage de paramètres (Valeurs par défaut)

Paramètres avec des valeurs par défaut :
Les paramètres par défaut doivent être déclarés en dernier pour éviter les confusions.

Exemple :

```
float MontantAvecTaxes(float mnt, float txtProv = 0.09975, float txtFed = 0.05);

/* ... */

resultat = MontantAvecTaxes (10.38); //calcul les taxes avec 0,09975 et 0,05

resultat = MontantAvecTaxes (10.38,0.04,0.06);

resultat = MontantAvecTaxes (10.38,0.08); // 0,05 devient le dernier paramètre

resultat = MontantAvecTaxes (); //erreur, il manque un paramètre
```

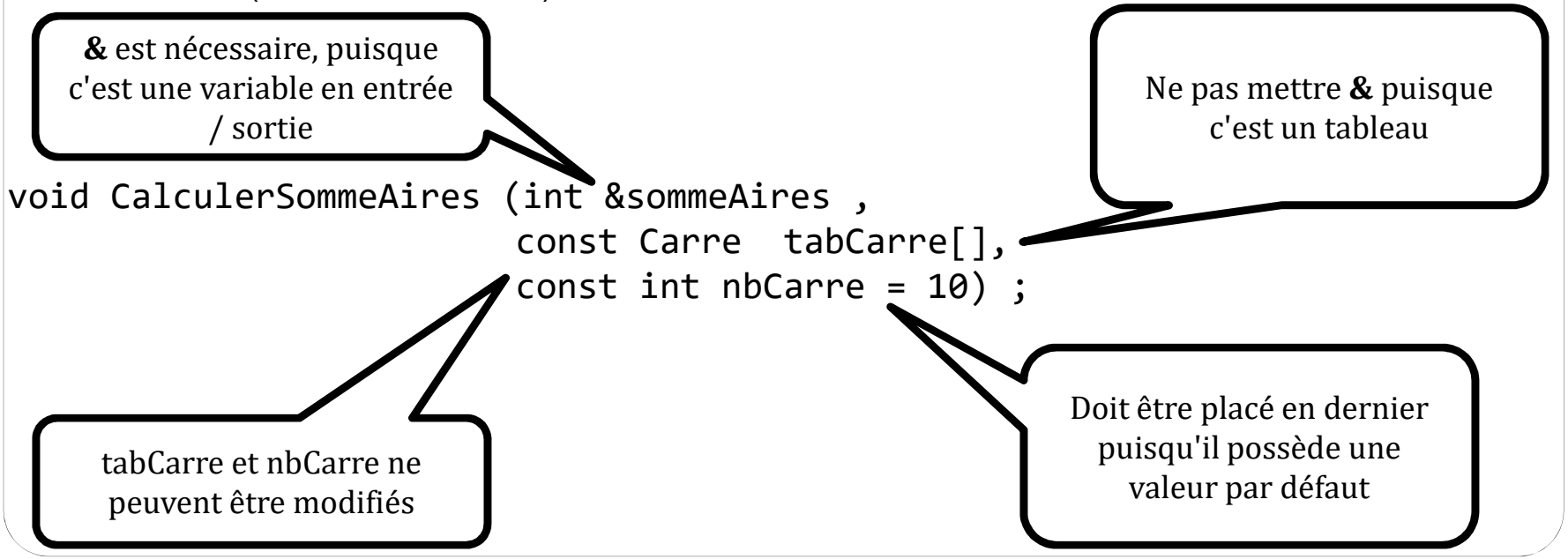
Passage de paramètres (exemple)

Écrire la déclaration d'une procédure **CalculerSommeAires** qui reçoit 3 paramètres :

tabCarre (en entrée) : un tableau d'objets de type Carre

nbCarre (en entrée) : le nombre de carrés dans le tableau (si la valeur n'est pas spécifiée, celle-ci doit être fixée à 10 – souvenez-vous qu'en C++, un tableau ne connait pas sa taille.)

sommeAires (en entrée / sortie): la somme des aires des carrées



420-V31-SF – Programmation de Jeux Vidéo III

Retour du main

fparadis@cegep-ste-foy.qc.ca

Retour du main

- Théoriquement, une application C++ peut retourner une valeur, qui peut-être utilisé par le système d'exploration utilisateur.
- Mais c’est optionnel; votre main peut être void (void main())
- Ou alors main a comme type de retour un int (int main())

Retour du main

- Un retour de 0 signifie un déroulement correct de l'application.

- Un retour autre que 0 (habituellement 1) signifie une fin de processus inhabituel.
  - *Différents codes peuvent avoir des significations diverses.*