

LES PROGRAMMES MODULAIRES DIVISER SON PROGRAMME EN FONCTIONS

Module 3

420-C2I-IN Programmation II

Godefroy Borduas – Automne 2021

1

QUEL EST L'OBJECTIF DU COURS ?

- Comprendre la notion de fonction
 - Utilité d'une fonction
- Comment créer une fonction ?
 - Définition des fonctions (prototype)
 - Déclaration des fonctions (corps)
- Comment utiliser une fonction ?
- Comment partager des informations avec les fonctions ?
- Petite attention au passage des paramètres et à la portée des variables
- Qu'est-ce qu'une bonne fonction ?

2

LA NOTION DE FONCTION

3

QU'EST-CE QU'UNE FONCTION ?

- Petit bout de code
- Réalise une opération concrète et définie
- Exemple :
 - Calculer la puissance de x par y
 - Calculer la différence de jour entre deux dates
 - Gérer un menu

4

POURQUOI UTILISER UNE FONCTION ?

- Réduire la taille du *main*
- Faciliter la lecture et la révision du code
- Réduis l'écriture de code

5

ALLONS-Y AVEC UN EXEMPLE DE PROGRAMME

- Créer un programme qui réalise le calcul x^y
- L'algorithme naïf est :

```
Resultat = 1
Pour i = 1, y, 1
    Resultat = Resultat * x
Écrire 'La puissance de' x 'est' Resultat
```
- Votre programme doit réaliser les étapes suivantes :
 Calculer 3^{16} ($x = 3, y = 16$)
 Calculer 42^{24} ($x = 42, y = 24$)
 Calculer 1048^{2048} ($x = 1048, y = 2048$)

INTERDIT !
D'utiliser la
méthode pow

6

QUELLE EST VOTRE CONCLUSION ?

- C'est long
- C'est répétitif
- Plusieurs sources d'erreur possible
 - Oublier de réinitialiser la variable Résultat
 - Oublier de spécifier la valeur x ou de y
 - Se tromper dans les variables
- C'est long pour corriger une erreur

7

LA SOLUTION

- Créer une fonction qui calcule la puissance
- Elle pourra être appelée à chaque besoin
- Les variables dans la fonction sont réinitialisées à chaque fois
- Besoin de changer ou de corriger ? Un seul endroit
- permet de partager sa méthode
 - On verra ce point plus tard

8

CRÉER SES FONCTIONS

Première étape : Déclarer sa fonction

9

QU'EST-CE QUE LA DÉCLARATION DE FONCTION ?

- Une annonce pour le compilateur
- Le compilateur réserve l'espace mémoire
 - Réserve le nom
 - Réserve l'espace des paramètres (variable d'entrée)
- S'appelle aussi un **prototype**

10

COMMENT ÉCRIRE UN PROTOTYPE

- Possède trois champs
 - Son type de retour (ce que la méthode renvoie)
 - Son nom (comment la différencier)
 - Les paramètres de la fonction (ce qu'elle reçoit pour fonctionner)
- Le prototype a toujours la même forme
`Type_de_retour nom_fonction(type para_1, type para_2, ..., type para_N);`
 - Le nom des paramètres est facultatif, seuls les types comptent
 - Le type **void** indique qu'aucune valeur ne sera retournée. Le retour est donc vide.
- Les prototypes sont toujours décrits **AVANT** le *main*

11

IMAGINONS LA MÉTHODE PUISSANCE

- Reprenons l'exemple du calcul de la puissance
- Avant d'écrire le prototype, il faut décrire son diagramme d'action
- La fonction d'un diagramme pour une méthode sera toujours comme suit :
`Type_retour Nom(paramètre1, paramètre2, ..., paramètreN)`
 Instructions
 retour Variable retournée
- Si la fonction n'a pas de paramètre d'entrée, on inscrit simple **void** dans les parenthèses

12

LE DIAGRAMME D'ACTION ET LA DÉCLARATION DE PUISSANCE

- En somme, le diagramme est :

```
double Puissance (x,y)
    Resultat = 1
    Pour i = 1, y, 1
        Resultat = Resultat * x
    retour Résultat
```

- Son prototype est donc :
`double Puissance(int x, int y);`
- Le prototype suivant est aussi valide :
`double Puissance(int, int);`

13

CRÉER SES FONCTIONS

Deuxième étape : Déclarer ses fonctions

14

SANS CORPS, UNE FONCTION N'EST PAS UTILISABLE

- La déclaration permet de créer la substance de la fonction
- On définit ses instructions (ce qu'elle doit faire)
- Une fonction sans corps n'est pas une fonction
- Les définitions sont toujours placées après le *main*
- Les fonctions ont toujours la même forme :

```
Type_de_retour nom_fonction(type para_1, type para_2, ..., type para_N)
{
    // Série d'instructions
    return Variable_a_retourner; // Instruction de retour
                                // (seulement pour les fonctions qui n'ont pas un type void)
}
```

15

UN LIEN ENTRE LA DÉFINITION ET LA PREMIÈRE LIGNE DE LA DÉCLARATION ?

- Certainement, la définition annonce la déclaration
- Les deux doivent correspondre
 - Le nom des paramètres doit être identique s'ils sont présents dans la définition
- Si la définition ne correspond pas à la première ligne de la déclaration
 - Le compilateur va refuser votre code
 - Le projet ne compilera pas

16

EXEMPLE DE LA MÉTHODE PUISSANCE

- Pour rappel, la déclaration est :
`double Puissance(int x, int y);`
- Par conséquent, sa définition est :
`double Puissance(int x, int y)`
`{`
`// Liste d'instruction`
`return Resultat;`
`}`
- Qu'est-ce qu'on met comme instruction ?

17

LES INSTRUCTIONS SERONT CELLES DE NOTRE DIAGRAMME

```
double Puissance(int x, int y)
{
    double Resultat = 1;
    for(int i = 1; i <= y; i++)
    {
        Resultat *= x;
    }
    return Resultat;
}
```

Note sur l'instruction de retour
Si l'instruction `return` n'a aucune valeur, alors elle met fin à la fonction sans rien retourner (valable pour les fonctions de type `void`).

18

UTILISER SES FONCTIONS

En gros... comme toutes les fonctions

19

L'APPEL D'UNE FONCTION EST SIMPLE

- Il suffit d'utiliser le nom de la fonction suivi des parenthèses

- Exemple d'une fonction sans paramètre

```
void exemple1(); // Définition
```

```
exemple1(); // Utilisation
```

- Exemple d'une fonction avec paramètre

```
void exemple2(int, int); // Définition
```

```
exemple2(42, 42); // Utilisation
```

- Si la fonction a un retour, on affecte la valeur à une fonction

```
int exemple3(int); // Définition
```

```
int n = exemple3(42); // Utilisation
```

Forme générale

```
nom_fonction(para1, para2, ..., paraN);
```

20

VOUS AVEZ DÉJÀ UTILISÉ DES FONCTIONS

- Regardez vos notes de C++

```
clrscr();
Rep = _getche();
Rep = MessageBoxA(NULL, "Voulez-vous continuer ?", "Système B11",
MB_YESNO);
```

21

DANS NOTRE CAS, NOTRE PROGRAMME DEVIENT...

```
#include <iostream>
using namespace std;
double Puissance(int, int);

int main()
{
    double p1 = Puissance(3, 16);
    cout << p1 << endl;
    double p2 = Puissance(42, 24);
    cout << p2 << endl;
    double p3 = Puissance(1024, 2048);

    cout << p3 << endl;
}

double Puissance(int x, int y)
{
    double Resultat = 1;
    for(int i = 1; i <= y; i++)
    {
        Resultat *= x;
    }
    return Resultat;
}
```

22

COMMENT PARTAGER DES INFORMATIONS AVEC LES FONCTIONS ?

23

PETIT QUIZ RAPIDE !

- Combien de paramètres peut-on transmettre à une fonction ?
 - 256 (est-ce qu'on va vraiment l'atteindre ?)
- Quel type puis-je transmettre dans une fonction ?
 - Tous les types incluant les types simples (int, char, float, ...), les types d'agrégats (string, int[], float[], ...), les types de structure et les pointeurs (notion du prochain cours)
- Combien de variable peut-il être renvoyé par une fonction ?
 - Une seule variable et le type doivent être définis à l'avance
- Quel type peut-être renvoyé par une fonction ?
 - Tous les types incluant les types simples (int, char, float, ...), les types d'agrégats (string, int[], float[], ...), les types de structure et les pointeurs (notion du prochain cours)

24

PETITE ATTENTION AU PASSAGE DES PARAMÈTRES ET À LA PORTÉE DES VARIABLES

25

COMMENT C++ TRANSFÈRE VOS VARIABLES ?

- C++ utilise le **passage par valeur**
- En gros, la variable en paramètre est **copiée** dans une nouvelle variable dite **locale**
 - Conséquence : Tous les calculs réalisés sur la variable locale n'affectent pas à la variable d'origine (celle qui a servi à l'appel de la fonction)
 - Conséquence II : Le programme doit utiliser le double d'espace pour la même valeur

26

QU'EST-CE QU'UNE VARIABLE LOCALE?

- Il s'agit d'une variable qui existe uniquement dans son **bloc de déclaration**
- Qu'est-ce qu'un **bloc de déclaration** ?
 - Il s'agit d'une suite d'instruction incluse entre des accolades { }.
 - Une fonction est un bloc de déclaration
 - La liste des instructions de **if** est un autre exemple de bloc
- En somme, la variable locale existe tant et aussi longtemps que nous sommes dans le bloc
 - Dès qu'on sort, la variable est détruite

27

EFFET DU PASSAGE DE VARIABLE

```
void Test(int);
```

```
int main() {
    int i = 2;
    Test(i);
    cout << i << endl;
}
```

Variable
locale à Main

```
void Test(int j) {
    j = 9;
    cout << j << endl;
    return;
}
```

Variable
locale à Test

- Affiche la valeur **9**
- Affiche la valeur **2**

28

IMPACT SUR LA MÉMOIRE

- Après la première ligne de main

```
int main() {
    int i = 2;
    Test(i);
    cout << i << endl;
}
```

Adresse	Valeur
0x...1	int i (2)
0x...2	
0x...3	
0x...4	
0x...5	
0x...6	
0x...7	
0x...8	
0x...9	

29

IMPACT SUR LA MÉMOIRE

- Après la **deuxième** ligne de main

```
int main() {
    int i = 2;
    Test(i);
    cout << i << endl;
}
void Test(int j) {
    j = 9;
    cout << j << endl;
    return;
}
```

Adresse	Valeur
0x...1	int i (2)
0x...2	
0x...3	
0x...4	
0x...5	
0x...6	int j (2)
0x...7	
0x...8	
0x...9	

30

IMPACT SUR LA MÉMOIRE

- Après la **première** ligne de **test**

```
int main() {
    int i = 2;
    Test(i);
    cout << i << endl;
}

void Test(int j) {
    j = 9;
    cout << j << endl;
    return;
}
```

Adresse	Valeur
0x...1	int i (2)
0x...2	
0x...3	
0x...4	
0x...5	
0x...6	int j (9)
0x...7	
0x...8	
0x...9	

31

IMPACT SUR LA MÉMOIRE

- Après la **troisième** ligne de **main**

```
int main() {
    int i = 2;
    Test(i);
    cout << i << endl;
}

void Test(int j) {
    j = 9;
    cout << j << endl;
    return;
}
```

Adresse	Valeur
0x...1	int i (2)
0x...2	
0x...3	
0x...4	
0x...5	
0x...6	
0x...7	
0x...8	
0x...9	

32

MAINTENANT, IMAGINER UN TABLEAU DE
STRING DE 256 CASES À TRANSFÉRER

Spoiler !
On voit une solution dès le
prochain cours.

33

QU'EST-CE QU'UNE BONNE
FONCTION ?

34

UNE BONNE FONCTION RESPECTE LE *CLEAN CODE*

1. Une fonction doit être courte
 - Maximum de 25 lignes de codes, si on en a besoin de plus, alors on parle d'une autre fonction.
2. La fonction n'a qu'une seule responsabilité et ne fait qu'une chose
 - La fonction ne doit faire qu'une seule tâche et non deux
3. Le nom de la fonction doit correspondre à ce que la fonction fait
 - Imaginer la fonction calcul avec le nom **Babloubabloublou**
4. Le bloc et les instructions doivent être indentés
 - Pitié mes yeux
5. Aucune duplication de code
 - Pourquoi se répéter ? On est des paresseux
6. Aucun effet secondaire (side effect)
 - La fonction ne doit pas modifier directement une valeur
7. Si c'est mort, ça dégage !
 - Ce n'est pas beau et pas lisible

C'est un critère
d'évaluation !

35

SURPRISE ! LA FONCTION RÉCURSIVE

36

QU'EST-CE QU'UNE FONCTION RÉCURSIVE ?

- Fonction dont le calcul nécessite l'appel d'elle-même
- Exemple : La suite de Fibonacci
 - Suite mathématique dont les valeurs dépendent des valeurs précédentes
 - 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144 ...
 - Défini mathématiquement par :

$$F_n = \begin{cases} 0, si\ n = 0 \\ 1, si\ n = 1 \\ F_{n-1} + F_{n-2}, si\ n > 1 \end{cases}$$

- Les fonctions récursives ont toujours un **point d'arrêt**
 - Sinon, on tombe dans une boucle infinie

37

EN TERMES DE CODE :

- Besoin d'un paramètre : **n** soit l'indice du nombre à calculer
- Retour la valeur de la suite à la position **n**
- La suite est décrite pour les entiers positifs seulement
- Le prototype est alors :
`unsigned int Fibonacci(unsigned int);`
- Le calcul est donc :
Si $n = 0$, retourne 0
Si $n = 1$, retourne 1
Sinon retourne $F(n - 1) + F(n - 2)$

<https://replit.com/@ColoneSaumon/Fibonacci>

38

QU'EST-CE QUI SE PASSE DANS UNE FONCTION SANS POINT D'ARRÊT

- La suite infinie de Godefroy

$$g_i = \begin{cases} \frac{n_{i-1}}{2}, & \text{si } n \text{ est pair} \\ 2n_{i-1}, & \text{si } n \text{ est impair} \\ 1000, & \text{si } n = 2 \\ -n_{i-1}, & \text{si } n < 0 \\ 500, & \text{si } n = 0 \end{cases}$$

- La suite ne peut jamais donner de résultat

<https://replit.com/@ColonelSaumon/SuiteInfiniDeGodefroy>

39

EXERCICES

40

SÉRIE D'EXERCICES

1. Écrivez une fonction **distance** ayant comme paramètres 4 doubles xa,ya et xb,yb qui représentent les coordonnées de deux points A et B et qui renvoient la distance AB. Tester cette fonction.
2. Reprenez la fonction précédente, mais avec la structure **Point**
3. Écrivez une fonction f ayant en paramètres un tableau t de taille quelconque et un entier n indiquant la taille du tableau. f doit renvoyer par un booléen indiquant s'il existe une valeur comprise entre 0 et 10 dans les n premières cases du tableau t. Tester cette fonction.

41

EXERCICE DE FONCTION RÉCURSIVE

1. Écrivez une fonction qui retourne la valeur de la factorielle de l'entier n transmit par paramètre.
 - La factorielle correspond à la multiplication de toutes les valeurs entiers entre 0 et n.
 - La factorielle de 0 est par définition 1.
2. Écrivez une fonction qui calcule la grande suite de Godefroy. Votre fonction doit retourner la valeur pour l'indice n. n est un entier transmis par paramètre à la fonction.
 - La grande suite de Godefroy est décrite comme suit :

$$G_n = \begin{cases} G_{-n}, & n < 0 \\ 0, & \text{si } n = 0 \\ i + 1, & \text{si } 1 \leq n \leq 9 \\ G_{i-1} + G_{i-2} + 2G_{i-3}, & \text{si } n \geq 10 \end{cases}$$

42