Cours de Langage C
Les fonctions

Programmation modulaire

- Un programme dépassant une ou deux pages est difficile à comprendre
- Une écriture modulaire permet de scinder le programme en plusieurs parties et sous-parties
- En C, le module se nomme la « fonction ».
- Le programme principal décrit essentiellement les enchaînements des fonctions

Programmation modulaire

- Bien différencier :

  Le texte (ou code) d'un programme

  qui est donc une suite de fonctions non emboîtées (on ne définit pas une fonction dans une autre fonction)

  →*Une fonction appelée dans une autre fonction a son code propre séparé de la fonction appelante*

  L'exécution d'un programme

  qui va enchaîner instructions, appels de fonctions (appelant elles-mêmes des fonctions) etc.

Les fonctions

- Dès qu'un groupe de lignes revient plusieurs fois on les regroupe dans une fonction
- Une fonction se reconnaît à ses ()
- Une fonction en C est assez proche de la notion mathématique de fonction:
- *Exemples* :

```
y = sqrt(x) ;
Z = pgcd(A,B) ;
```

Intérêt des fonctions

- Lisibilité du code
- Réutilisation de la fonction
- Tests facilités
- Évolutivité du code
- *Plus tard* : les fonctions dans des fichiers séparés du main.c
- *Nb : une fonction peut faire appel à d'autres fonctions*
  - *dans son code*
  - *dans ses arguments*

Bibliothèques de fonctions

- Il existe des bibliothèques de fonctions déjà programmées.
- *Exemples* :
  - math.h : *fonctions math.*
  - stdio.h: *standard input-output*
  - stdlib.h : *bibli. standard*
  - time.h : *fonctions temporelles*

→ *Nous créerons nos propres bibliothèques au cours de la session de C*

2 types de fonctions

- Des fonctions qui s'exécutent sans retourner de valeurs

  → *nommées procédures dans certains langages*

  →Seront typées **void**

  Ex : une fonction qui affiche « bonjour »

```
void affiche_bonjour()
{
printf(" bonjour ");
}
```

- Des fonctions qui s'exécutent et retournent une valeur

  Exemples : sin(x) ; z = sqrt(x) ;

  → Auront le type de la valeur à retourner

Définition, déclaration, et appel d'une fonction

- On rencontre le nom des fonctions dans 3 cas :
  - **Déclaration** : le type de la fonction et de ses arguments

    → 1 seule fois
  - **Définition** : codage de la fonction

    → 1 seule fois
  - **Appels** (= utilisations) de la fonction

    → n fois

Paramètres réels – paramètres formels

- Un paramètre ou argument **réel**, est une valeur ou une variable qui est mis entre parenthèses lors de l'appel de la fonction.
  - Il existe vraiment en mémoire.
- Un paramètre ou argument **formel** est un nom de variable utilisé lors de la déclaration de la fonction.
  - Le nom peut être omis (pas conseillé)
  - Ne correspond pas à un emplacement mémoire

Déclaration d'une fonction

- Permet au compilateur de vérifier l'adéquation des types et de réserver l'espace mémoire pour la valeur de retour
- A l'aide d'un **prototype** de fonction utilisant des *paramètres formels typés de la forme :*

Type-retourné **NOM-FONCTION (type1 paramètre1, type2 paramètre2, …) ;**

double calculePrixNet(**double prix**, **double tauxTVA**)

NB : on peut définir une fonction avec autant de paramètres formels qu'on veut. Dans l'exemple, il y a deux paramètres formels.

Définition d'une fonction

- C'est le **code** de la fonction, de la forme :

Type-retourné NOM-FONCTION (type1 paramètre1, type2 paramètre2, …)

{

Déclaration des autres variables de la fonction;

Code de la fonction;

return (valeur-de-la-fonction) ;

}

Attention !

- En C, une fonction ne peut retourner qu'une valeur (au plus) grâce à la commande *return*
- Le type de la fonction doit être le même que celui de la valeur retournée
- Le programme appelant doit stocker ce résultat dans une variable de même type (ou bien ne rien stocker)
- Quand une fonction ne retourne pas de valeur elle est typée *void*
- *Exemples* : int saisirInt() ; void afficheBonjour();

Le return

- Retourne la valeur au programme appelant

- Et interrompt immédiatement l'exécution de la fonction
  - On peut avoir plusieurs *return*
  - Mais un seul *return* pris en compte à chaque exécution

Variables locales – variables globales

- Définition : Un bloc est la partie de code compris entre {}
- Une variable créée dans un bloc n'existe que dans ce bloc
  - C'est une **variable locale** au bloc
  - Elle ne sera pas connue en dehors
  - Sa valeur est perdue à la sortie du bloc
  - *« Sa durée de vie est celle du bloc »*

Variables locales – variables globales

- Une variable **globale** existe en dehors de tout bloc
- Elle a sa mémoire réservée pour toute l'exécution du programme
- « Sa durée de vie est celle du programme »
- *Exemple :*

```
int i ;
main()
{
    i = 2;
    printf("%d", i);
}
```

- ***Conseil : Soyez le plus local possible***

Déclaration de variables dans les fonctions

- De 2 manières :

```
int triple (int x )
{
int y ;
y = 3 * x ;
return (y) ;
}
```

→ x est **locale** à la fonction
→ Elle est initialisée lors de l'appel à la valeur fournie par le programme appelant
→ Sa valeur sera perdue à la sortie de la fonction
→ *Nb : on parle de passage par valeur des arguments :* ***leurs valeurs sont recopiées dans des variables locales à la fonction***
→ y est **locale** à la fonction
→ Sa valeur sera perdue à la sortie de la fonction

Appels de fonctions : exemples

- Utilisations :

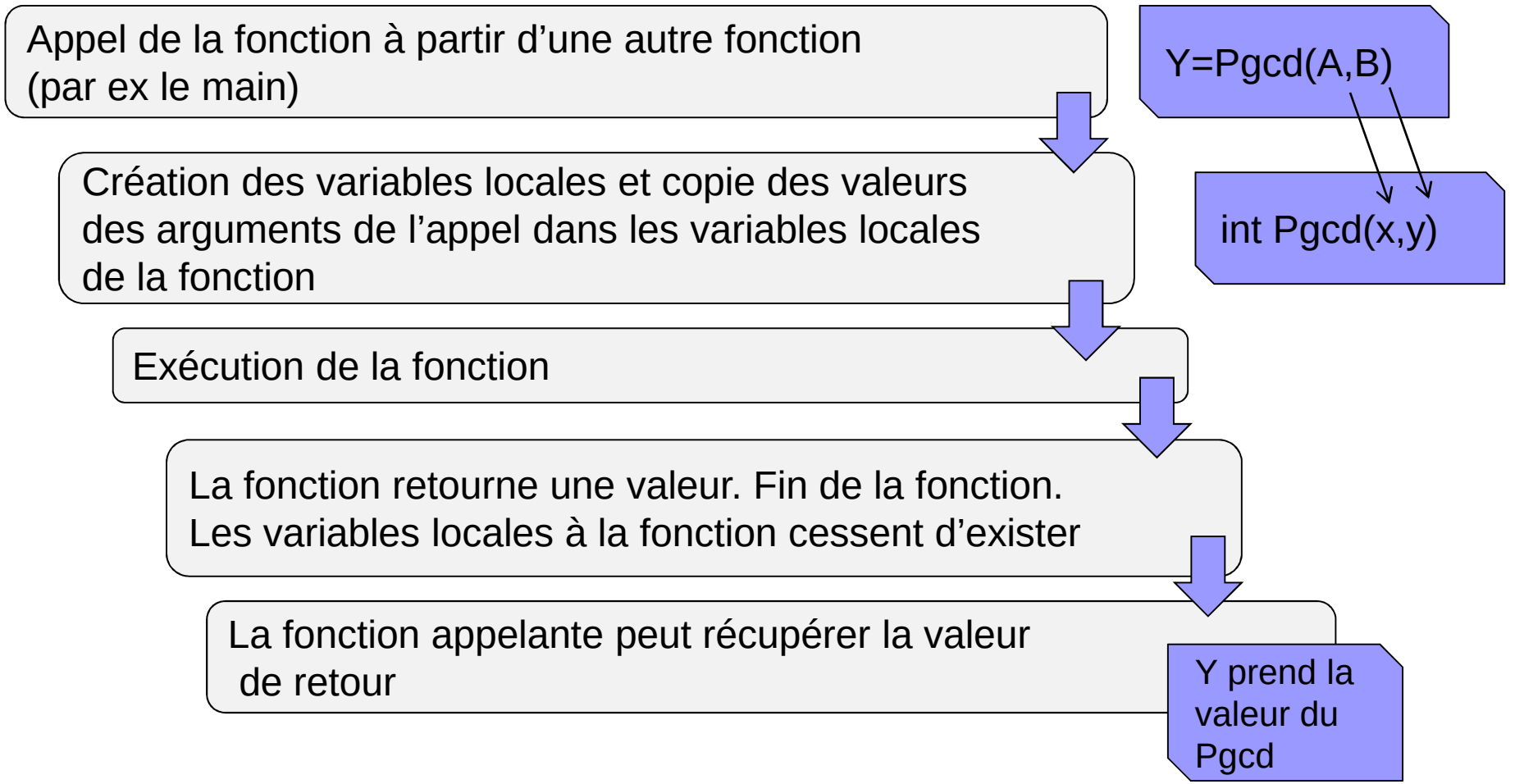
```
int main()
{
   int a = 2 ;
   int b ;
   triple(2) ;
   triple(a) ;
   b = triple(a) ;
   a = triple(a) ;
   return 0;
}
```

→ la valeur de retour est perdue

→ la valeur de retour est sauvée dans b

→ la valeur de retour est sauvée dans a

Appel d'une fonction



Fonctions et tableau

- Un tableau peut être un argument d'entrée d'une fonction
  - Mais pas un élément retourné (pas à ce stade)
- La syntaxe est :

```
int tab[22] ; int n = 22 ;
… // bout de code
m = moyenne(tab,n) ;
… // bout de code
x = maximum(tab,n)) ;
```

- On transmet donc le nom du tableau sans crochets
  - Très souvent, le nombre d'éléments du tableau sur lequel on souhaite travaillé est aussi un argument de la fonction pour donner un caractère générique à la fonction.

D’autres exemples de fonctions

aireRectangle(a, b)

moyenne(a, b, c, 2, 18, 9)

mensualite(sommeEmprunt, tauxEmprunt, nbreDeMois)

afficher(aireRectangle(a,b))

- Une fonction peut avoir 1 ou plusieurs paramètres d’entrée
- Les paramètres peuvent être de types différents
- Une fonction peut utiliser comme argument une autre fonction

Structure d'un code utilisant des fonctions

1

```
#include …
#define …
```

2

Déclarations des fonctions (prototypes)

3

```
main()
{
    …..
    appels aux fonctions
    …..
}
```

4

Définitions des fonctions

Codage d'une fonction : exemple

1

2

3

4

```
#include <stdio.h>

int triple(int x) ; //prototype

int main()
{
int a=2 ;
triple(2) ; //appels
triple(a) ;
a = triple(a) ;
return 0;
}

int triple(int x) //définition
{
return (3*x) ;
}
```

La fonction *main()*

- main est une fonction
  - main()
  - int main()
  - void main()

  → Apparition possible de w*arnings* à la compilation

Les bonnes pratiques de programmation

- Une fonction ne fait en général qu'une chose
- Le nom de la fonction décrit cette chose ;
- Prendre le temps de bien choisir les fonctions, leur nom, leurs paramètres

  → Bien choisir un nom explicite … et l'utiliser par copier-coller avec son jeu de paramètres
- Une fonction reçoit un nombre limité de paramètre (2-3 dans la plupart des cas) ;
- Une fonction ne compte pas trop de lignes
- Tester chaque fonction avant de passer à l'écriture de la suivante

Les erreurs courantes avec les fonctions

- Une fonction est déclarée mais non définie
- Une fonction est appelée et n'existe pas
- Le type de la fonction ne correspond pas au type de la valeur retournée
- La valeur retournée n'est pas stockée dans une variable du bon type
- Entre la déclaration, la définition et l'appel, le nombre de paramètres n'est pas le même
- Au moins un paramètre n'a pas le bon type
- Ne confondez pas *valeur retournée* par la fonction (qui peut être stockée dans une variable en mémoire) et *affichage à l'écran d'un résultat* (qui n'est pas automatiquement stocké en mémoire)

En-tête imposé de fonction

```
/*
Role de AireRectangle: Calcul de l'aire d'un rectangle
Entrees :  a,b : Largeur et longueur
Sortie :  l'aire du rectangle
Auteur : Adan Dejour – le 25-12-2014
*/
double aireRectangle(double a, double b)
{
   …
}
```

Si vous réutilisez une fonction de qqn d'autre, vous devez l'indiquer.

Conseils

- Si vous utilisez beaucoup de fonctions, tenez leur liste à jour (Tableur, texte, …)
- Lorsque vous écrivez une fonction : testez-la et assurez-vous de son bon fonctionnement avant de passer à l'écriture de la suivante !!
- Ce qu'on ne doit **jamais** faire : écrire toutes les fonctions et tester ensuite tout d'un bloc.
- Evitez les printf dans une fonction qui n'est pas dédiée à l'affichage. Vous pouvez utiliser des affichages avec printf pour les débugger, mais retirez-les dès que la fonction marche correctement.

Au final

- Au niveau du texte :
  - Un programme en C est un ensemble disjoint de fonctions dont une seule porte le nom de **main (programme principal)** et constitue le point d'entrée du programme.
  - On verra qu'on peut répartir les fonctions dans plusieurs fichiers textes
- Au niveau de l'exécution :
  - Un programme en C est une succession d'appels d'instructions et de fonctions pouvant utiliser comme paramètres des résultats de fonctions (et ainsi de suite).