Le langage C

Références

- Livre
  - Méthodologie de la programmation en C : Bibliothèque standard – API Posix. Jean-Pierre Braquelaire. 3ème édition Dunod
  - Le langage C. B.W. Kernighan et D.M. Ritchie. 2ème édition, MASSON.
- Cours en ligne sur internet
  - www.developpez.com

Plan

- Analyse descendante
- Sous-programme et fonctions
- Fonctions en C

Analyse descendante

- Problématique

Problème : Comment traiter les problèmes trop complexes pour être appréhendés en un seul bloc (dans la fonction main par exemple) ?

Solution : On peut appliquer la méthode "Diviser pour régner", et décomposer le problème en sous-problèmes plus simples : c’est l’analyse descendante

Analyse descendante

- Principe d'analyse descendante

Rôle : Méthode pour écrire un programme de qualité : lecture plus facile de programme, modularité du code

Principe :

- Abstraire

Repoussé le plus loin possible l'écriture de l'algorithme (codage)

- Décomposer

Décomposer la résolution du problème initial en une suite de "sous problèmes" que l'on considère comme résolus

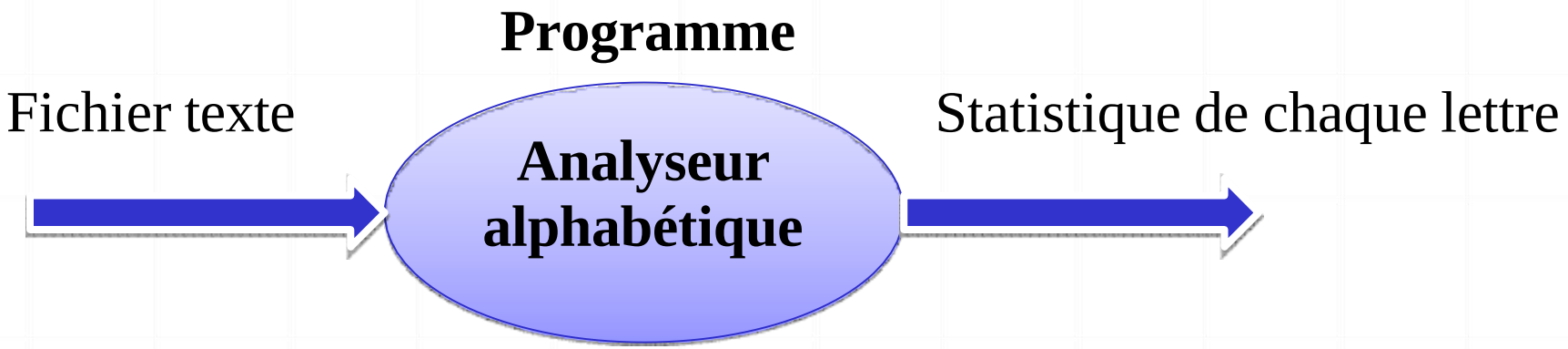
- Combiner

Résoudre le problème initial par combinaison des abstractions des "sous-problèmes"

Analyse descendante

- Exemple

Analyseur alphabétique : Écrire un programme analysant un fichier texte et indiquant le nombre d'occurrence de chaque lettre de l'alphabet dans le fichier



Analyse descendante

- Exemple
- Résoudre le problème revient à :

  Extraire un mot du fichier

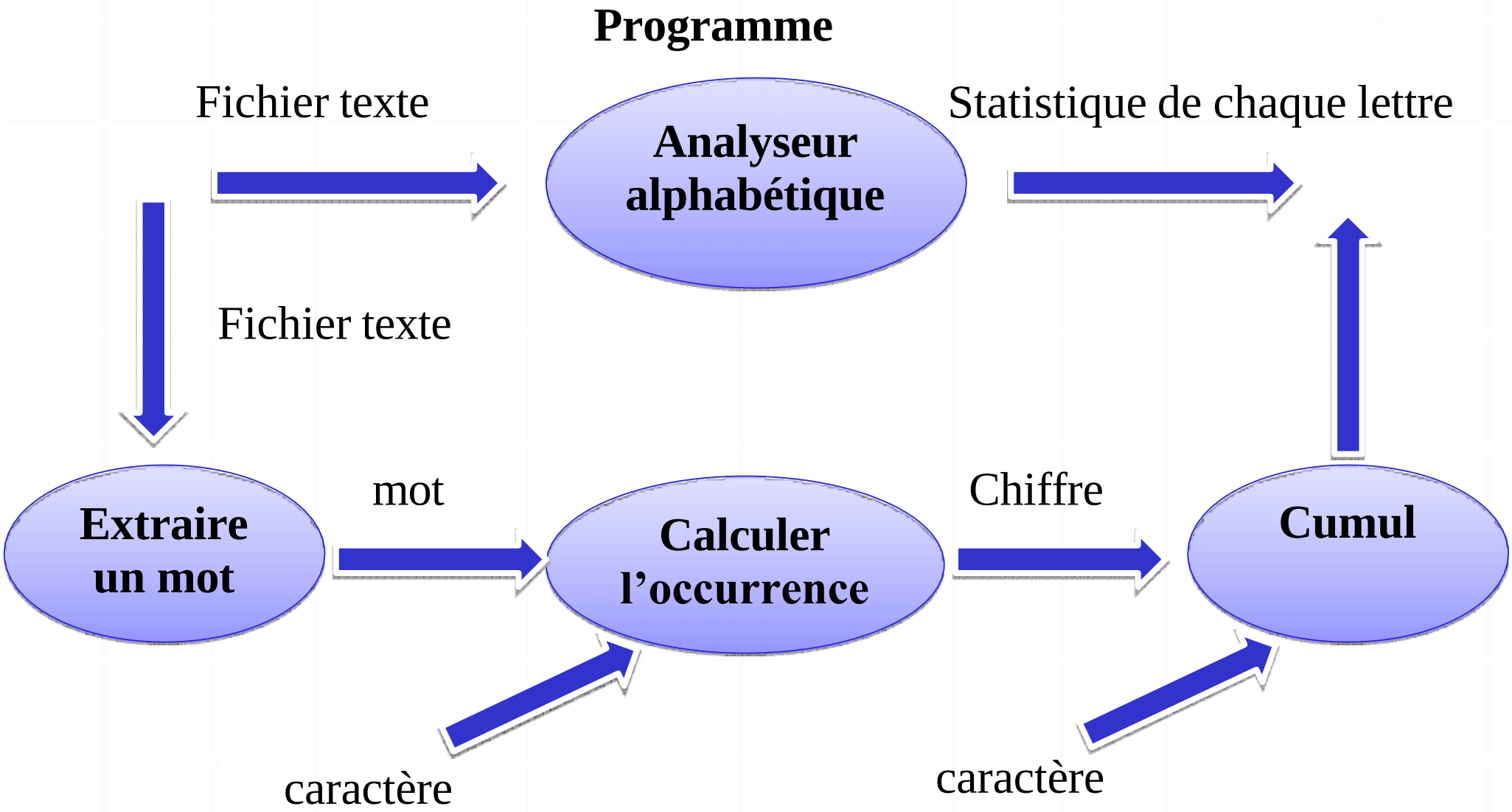
  Pour chaque, lettre de l'alphabet, compter le nombre d'occurrence

  Calculer le cumul d'occurrence de chaque lettre

  Répéter la traitement jusqu'au dernier mots du fichier.
- Chacun de ces sous-problèmes devient un nouveau problème à résoudre
- Si on considère que l'on sait résoudre ces sous-problèmes, alors on sait "quasiment" résoudre le problème initial

Analyse descendante

- Exemple



Fonctions et sous-programmes

- Sous-programmes et fonctions

  Donc écrire un programme qui résout un problème revient toujours à écrire des sous-programmes qui résolvent des sous parties du problème initial. Il faut comprendre les mots “programme” et de “sous-programme” comme “programme algorithmique” indépendant de toute implantation

- En algorithmique il existe deux types de sous-programmes :
  - Les fonctions : réalisent des traitements en se servant des valeurs de ce certaines variables et renvoient un résultat. Ils se comportent comme des fonctions mathématiques : y=f(x, y,. . .)
  - Les procédures : réalisent seulement des traitements mais ne renvoies aucun résultat

Fonctions et sous-programmes

- Fonctions et procédures

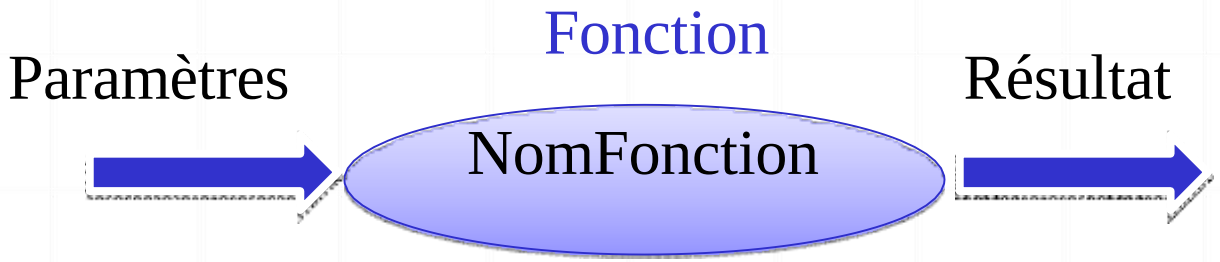
Les **fonctions** et les **procédures** sont des groupes d'instructions indépendants désignés par un nom. Elles ont plusieurs **intérêts** :

- permettent de **"factoriser" les programmes**, c-à-d de mettre en commun les parties qui se répètent
- permettent une **structuration** et une **meilleure lisibilité** des programmes
- **facilitent la maintenance** du code (il suffit de modifier une seule fois)
- ces procédures et fonctions peuvent éventuellement être **réutilisées** dans d'autres programmes

Fonctions en C

- Fonctions en C
- En C, il n'y a pas de différence entre fonctions et procédures :
  - il n'y a que des fonctions censées renvoyer une valeur d'un type spécifié
  - Un sous-programme (procédure) est une fonction qui ne renvoie rien (type void)

- Une fonction est un bloc d'instructions ayant :
  - Un type pour les valeurs qu'elle retourne (type du résultat)
  - Un nom (nom de la fonction)
  - Une liste de paramètres typés, entre parenthèses (paramètres



Fonctions en C

- Exemple

La fonction SommeCarre suivante calcule la somme des carrées de deux réels x et y :

```
float SommeCarre (float x, float y )
{
      float z;
      z  = x*x+y*y;
      return z;              // on retourne le résultat à l'aide de l'instruction return
}
```

La fonction Pair suivante détermine si un nombre est pair :

```
int Pair(int x)
{
      if (x%2 == 0)
            return 1;        // vue qu'il n'y a pas de type booléen en C, on retourne 1
      else                   //pour dire que c'est vrai et 0 pour faux
            return 0;
}
```

Fonctions en C

- Exemple

La fonction Affiche permet d'afficher le message Bonjour à l'écran:

```
void  Affiche ( )
{
      printf("Bonjour \n");
/* L'affichage à l'écran n'est pas considéré comme un résultat en C, donc pas
besoin de l'instruction return */
}
```

La fonction afficheInt permet d'afficher le contenu d'une variable entière à l'écran:

```
void  AfficheInt ( int i )
{
      printf("La valeur de i est %d \n", i );
/* L'affichage à l'écran n'est pas considéré comme un résultat en C, donc pas
besoin de l'instruction return */
}
```

Fonctions en C

- Exemple

La fonction ValeurAbsolue retourne la valeur absolue d'un entier passé en paramètre:

```
int ValeurAbsolue ( int X )
{
        int valeur;
        if (X > =0 )
                valeur = X;
        else
                valeur = - X;

        return valeur;
/*  remarquer que le type de la variable valeur est le même que celui du retour
de la fonction ValeurAbsolue
*/
}
```

Fonctions en C

- Remarque

Les paramètres des fonctions sont vus comme des variables locales au bloc de code de la fonction. On peut définir d'autres variables dans le bloc.

Si la fonction ne renvoie aucun résultat, on utilise la mot réservé void. L'affichage à l'écran n'est pas considéré comme un un résultat (un retour)

Si la fonction n'a besoin d'aucun paramètre on écrit simplement ( ) ou (void)

```
int ValeurAbsolue ( int X )
{
        int valeur;
        if (X > =0 )
                valeur = X;
        else
                valeur = - X;

        return valeur;

}
```

Fonctions en C

- Instruction return

  On spécifie la valeur que renvoie une fonction au moyen de l'instruction **return**

  - **Valeur de même type que le type de retour déclaré de la fonction**

```
int ValeurAbsolue ( int X )  // retourne la valeur absolue d'un entier passé en paramètre
{
      int valeur;
      if (X > =0 )
            valeur = X;
      else
            valeur = - X;

      return valeur;
/*  remarquer que le type de la variable valeur est le même que celui du retour de la fonction
ValeurAbsolue
*/
}
```

Fonctions en C

- Instruction return

  L'instruction return permet la terminaison anticipée de la fonction

  - **Peut exister en plusieurs exemplaires**

```
int Pair(int x)
{
	if (x%2 == 0)
		return 1;	// vue qu'il n'y a pas de type booléen en C, on retourne 1
	else			//pour dire que c'est vrai et 0 pour faux
		return 0;

	printf ("Ce message ne sera jamais affiché \n");
}
```

Fonctions en C

- Instruction return

Pour les fonctions ne retournant rien, soit on utilise l'instruction return sans argument, soit on met rien. Dans ce cas, le type de retour de la fonction est void

```
void  Affiche ( )
{
      printf("Bonjour \n");
}
```

```
void  Affiche ( )
{
      printf("Bonjour \n");

      return;              // optionnel
}
```

Fonctions en C

- Exercice

  Ecrire une fonction permettant de retourner le max de deux réels.

```
float   max2 ( float a, float b )
{
      float max;          // le résultat final sera stocké dans cette variable

      if (a > = b)
            max = a;
      else
            max = b;

      return max;
}
.
```

Fonctions en C

- Exercice

  Ecrire une fonction permettant de retourner le factoriel d'un entier donné

```
int   factoriel ( int n )
{
      int fac;          // le résultat final sera stocké dans cette variable
      int i;            // utilisée dans la boucle

      fac = 1;          // on initialise la variable à 1 pour pouvoir calculer les produits

      for (i =1; i < = n; i++)
      {
            fac = fac * i;              // ou encore fac *= i;
      }

      return fac;
}
```

Fonctions en C

- Fonction main : fonction principale

int main () est une fonction particulier qui retourne un entier et dont la liste des paramètres est vide. Elle est appelée la fonction principale.

Tout programme doit contenir obligatoirement la fonction principale, qui est exécutée lorsque le programme est lancé.

```
int   main ( )
{
      …

      return EXIT_SUCCESS;
}
.
```

Fonctions en C

- Exercice

Ecrire une fonction permettant d'afficher le contenu d'un tableau d'entiers passé en paramètre. La taille du tableau est aussi passé en paramètre de la fonction

```
void   afficheTableau ( int tab[], int n )
{
      int i;              // utilisée dans la boucle

      for (i =0; i <  n; i++)
            printf("la valeur de la case d'indice %d est : %d", i, tab[i]);
}
.
```

Fonctions en C

- Appel de fonction

  On appelle une fonction en donnant son nom, suivi de la valeur des paramètres de l'appel, dans l'ordre de définition

  Le nom de fonction avec ses arguments est une expression typée utilisable normalement

```
int ValeurAbsolue ( int X )
{
      int valeur;
      if (X > =0 )
            valeur = X;
      else
            valeur = - X;

      return valeur;
}
```

```
int main ( )
{
      int val1;
      val1 = ValeurAbsolue (3);        ⟸  Appel de Fonction

      printf("la valeur absolue est : %d ", val1);

      return EXIT_SUCCESS;
}
```

Fonctions en C

- Appel de fonction

```
int ValeurAbsolue ( int X )
{
	int valeur;
	if (X > =0 )
		valeur = X;
	else
		valeur = - X;

	return valeur;
}
```

```
int main ( )
{
	int val1, val2;

	printf("Saisir un entier");
	scanf ("%d", &val1);

	val2 = ValeurAbsolue(val1);
	printf("la valeur absolue de %d est : %d ", val1, val2);

	return EXIT_SUCCESS;
}
```

Fonctions en C

- Appel de fonction

```
int ValeurAbsolue ( int X )
{
        int valeur;
        if (X > =0 )
                valeur = X;
        else
                valeur = - X;

        return valeur;
}
```

```
void  AfficheInt ( int i )
{
      printf("La valeur de i est %d \n", i );
}
```

```
int main ( )
{
        int val1, val2;

        printf("Saisir un entier");
        scanf ("%d", &val1);

        val2 = ValeurAbsolue(val1);
        AfficheInt(val2);

        return EXIT_SUCCESS;
}
```

Fonctions en C

- Exercice

Dans un exercice, nous avons défini une fonction qui permet de calculer le max de deux réel. Maintenant, on veut écrire une fonction max3 qui calcule de max de trois réels. Donner le code de max3 , en faisant un appel à la fonction max2

```
float   max2 ( float a, float b )
{
      int max;

      if (a > = b)
            max = a;
      else
            max = b;

      return max;
}
.
```

```
float   max3 ( float a, float b, float c )
{
      int m, max;

      m = max2 (a,b);

      max = max2(m,c)

      return max;
}
.
```

Fonctions en C

- Exercice

  Ecrire un programme qui demande à l'utilisateur de saisir trois réels et qui affiche leur max

Fonctions en C

- Corrigé

```
/* Fichier maximum.c
*  Ce fichier contient le programme qui
*  calcule le max de trois entiers
*  Auteur : M.HMAMOU
*  Version : v1
* Date création : aujourd'hui
*/
#include < stdlib.h>
#include < stdio.h>

/* Fonction qui calcule le max de 2 réels
float   max2 ( float a, float b )
{
      int max;
      if (a > = b)
            max = a;
      else
            max = b;
      return max;
}

/* Fonction qui calcule le max de  3 réels */
float   max3 ( float a, float b, float c )
{
      int m, max;
      m = max2 (a,b);
      max = max2(m,c)
      return max;
}

/* Fonction principale qui lance  l'exécution */
int   main ( )
{
      float x, y, z, max ;
      printf("Saisir trois réel : ");
      scanf ("%f%f%f",&x,&y,&z);

      max =  max3(x,y,z);
       printf("La max est %f : ", max);
      return EXIT_SUCCESS;
}
```