Les fonctions

EL Moukhtar ZEMMOURI

ENSAM – Meknès

Version 1.0 – 2015 / 2016

Sommaire

Introduction

- Un programme C peut être décomposé en sous programmes appelés fonctions.
- Pourquoi faire :
  - Facilite la résolution de problèmes complexes
  - Réutilisabilité du code
  - Vérification et maintenance de code
  - ...
- En C tout est fonction
  - C'est pourquoi on doit avoir au moins la fonction principale main.

Définition d'une fonction en C

- La définition d'une fonction est donnée par le code source de son algorithme
- Syntaxe :

```
type  nom_fonction (type1 arg1, ... , typeN argN) {
    Déclaration de variables locales
    Instructions
}
```

- Exemple : la fonction PGCD

```
int pgcd (int a, int b){
    int r;

    r = a%b;
    while (r != 0){
        a = b;
        b = r;
        r = a%b;
    }
    return b
}
```

Définition d'une fonction en C

- L'entête de la fonction contient :
  - Le type de la fonction : c'est le type de la valeur de retour de la fonction
    - Une fonction qui ne retourne rien est de type **`void`**
  - nom_fonction : c'est l'identificateur de la fonction
  - arg1, arg2, ... , argN : sont les arguments de la fonction
    - Appelés aussi paramètres formels
    - Si la fonction n'a pas d'arguments :

      `type nom_fonction ( )` ou **`type nom_fonction (void)`**

Définition d'une fonction en C

- Valeur de retour d'une fonction :
  - Une fonction renvoie une valeur à l'aide de l'instruction **`return`**
  - Syntaxe : **`return expression ;`**
    - C'est l'instruction de retour à la fonction appelante
  - Une fonction contient une ou plusieurs instructions return
  - La première return exécutée met fin à la fonction et renvoie une valeur à la fonction appelante
  - Une fonction de type void :
    - Se termine par **`return ;`**
    - Ou pas d'instruction return.

Exemple

```c
int prime (int n){
    int i;

    if (n == 1) return 0;

    for (i=2; i<=n/2; i++)
        if (n%i == 0) return 0;

    return 1;
}
```

```c
void printPrimes(int n, int m){
    int p;

    for (p = n; p<=m; p++)
        if (prime(p))
            printf("%d\n", p);

    return;
}
```

- La fonction prime :
  - Si n == 1, la fonction se termine et revoie 0 (false)
  - Si on trouve un diviseur de n, la fonction se termine et renvoie 0
  - Sinon à la fin la fonction renvoie 1
- La fonction printPrimes :
  - L'instruction return à la fin est optionnelle.

Appel d'une fonction

- Une fonction peut être appelée par une autre fonction à l'aide de la syntaxe suivante :

  nom_fonction (param1, ... , paramN)

  - param1, ... , paramN : sont les paramètres éffectifs de la fonction
  - ➔ les valeurs sur lesquelles s'exécutera la fonction.
- L'ordre et les types des paramètres effectifs doivent concorder avec ceux des paramètres formels (arguments).

```c
int main()
{
    int n, m, p;

    n = 123;
    m = 1234;

    p = pgcd(n, m);
    printf("PGCD de %d et %d est %d\n", n, m, p);
    printf("PGCD de %d et %d est %d\n", n, m, pgcd(n, m));

    if (prime(n)) printf("%d est premier\n", m);
    else printf("%d n'est pas premier\n", m);

    printf("Nombres premiers entre %d et %d : \n", n, m);
    printPrimes(n, m);

    return 0;
}
```

Déclaration de fonctions

- Une fonction est déclarée par son prototype :

```
type  nom_fonction (type1, ... , typeN);
```

- Règle :
  - Toute fonction doit être définie ou déclarée avant sont premier appel.
- Exemple :

```
#include <stdio.h>

int pgcd(int, int);
int prime(int);
void printPrimes(int, int);
```

- Note :
  - Les noms des paramètres sont optionnels dans la déclaration.

Variables locales/globales

- Variable globale : une variable déclarée en dehors de toute fonction
  - ➔ elle est permanente
  - ➔ elle est accessible (visible) dans la partie du programme qui suit sa déclaration
  - ➔ par défaut initialisée à zéro.
- Variable locale : une variable déclarée dans une fonction
  - ➔ elle est temporaire
  - ➔ elle est accessible juste dans la fonction où elle est déclarée.
- Les paramètres sont traités comme des variables locales

Variables locales/globales

```c
#include <stdio.h>

int n;

void test(){
    n++;
    printf("appel numéro : %d\n", n);
}

int main(){
    int i;

    for (i=1; i<=10; i++)
        test();
}
```

```c
#include <stdio.h>

int n = 5;

void test(){
    int n = 0;
    n++;
    printf("appel numéro : %d\n", n);
}

int main(){
    int i;

    for (i=1; i<=10; i++)
        test();
}
```


Récursivité

- Exemple : la fonction factorielle

Version itérative

```c
int fact (int n){
    int f, i;

    f  = 1;
    for (i=2; i<=n; i++)
        f *= i;

    return f;
}
```

Version récursive

```c
int fact_rec (int n){
    if (n == 0 || n == 1) return 1;
    else return n*fact_rec(n-1);
}
```

Récursivité

- Une fonction récursive est une fonction qui appel elle même.
- Conception d'une fonction récursive :
  - Attention à la condition de terminaison