Procédures et fonctions

Mercredi 25 Octobre 2017

Michael FRANÇOIS

francois@esiea.fr

esiea
ECOLE D'INGENIEURS
DU MONDE NUMERIQUE

Objectif de ce cours

- Écrire un code modulaire via l'utilisation de procédures et fonctions.
- Comprendre les règles pratiques de l'utilisation des procédures et fonctions.
- Voir quelques exemples pratiques.

Pourquoi utiliser des fonctions ?

Il est important d'utiliser les fonctions pour :

1. factoriser le code, afin de le rendre plus lisible ;
2. centraliser la correction des bugs ;
3. mieux structurer le programme.

Fonction

- Une fonction est une portion de code isolé et dédié à la réalisation d'un calcul ou d'une tâche spécifique et qui **RENVOIE** un résultat.

- Exemple :

```
int getInteger() {
   int val;
   if (scanf("%d", &val) != 1) {
      printf("Invalid value!\n");
      exit(1);
   }
   return val;
}
```

Procédure

- Une procédure est une portion de code isolé et dédié à la réalisation d'une tâche spécifique et qui **NE RENVOIE PAS** de résultat.

- Exemple :

```
void Pb(){
   //ACTION
   printf("Il y a un bug!\n");
   exit(1);
}
```

Fonction et procédure sont définies en pratique par :

- Leur prototype qui donne les informations générales :

  - le nom de la fonction ;
  - les paramètres d'entrées (i.e. les informations dont la fonction a besoin pour faire son calcul ou sa tâche) ;
  - le type du résultat de l'exécution du code de la fonction ou void si c'est une procédure.

```
int sum(int var1, int var2) {  // Prototype
```

- Le bloc de code de la fonction (*i.e.* corps de la fonction).

Le bloc de code de la fonction :

- commence par déclarer toutes les variables locales (i.e. les variables dont on a besoin pour faire le calcul)
- enchaîne par la partie algorithmique (suite des instructions)
- contient (au moins) une fois l'instruction `return` pour retourner le résultat s'il s'agit d'une fonction.

```
int getInteger() {
   int val;
   if (scanf("%d", &val) != 1) {
      printf("Invalid value!\n");
      exit(1);
   }
   return val;
}
```

Quelques règles

• Pas de limite sur le nombre de fonctions (blocs de factorisation de code).

• Privilégier la spécialisation des fonctions (i.e. une tâche = une fonction).

• Aucune fonction ne peut contenir une autre fonction : ça n'a aucun sens car ça revient à mettre une tâche dans une autre.

- Une fonction (ou procédure) peut utiliser le résultat d'une autre fonction (ou une procédure) dans son bloc de code (appel de fonction ou de procédure)

- Une procédure doit être définie avant d'être appelée.

```c
int getInteger() {
   int val;
   if (scanf("%d", &val) != 1) {
      Pb();
   }
   return val;
}
void Pb(){
   printf("Invalid value!\n");
   exit(1);
}
//BAD: Pb() unknown when called
```

```c
void Pb(){
   printf("Invalid value!\n");
   exit(1);
}
int getInteger() {
   int val;
   if (scanf("%d", &val) != 1) {
      Pb();
   }
   return val;
}
//GOOD :)
```

Architecture d'un programme

```c
#include <stdio.h>
#include <stdlib.h>
```

```c
int getInteger() {
   //DECLARATION
   ...
   //ACTION
   ...
   //RETURN
}
```

```c
void Pb(){
   //ACTION
   printf("Il y a un bug!\n");
   exit(1);
}
```

```c
int main() {
   //DECLARATION
   ...
   //ACTION
   ...
   return 0;
}
```

Un programme en C :

- commence par les commandes pré-processeur (commençant par #)
- enchaîne par une succession de définitions de fonctions (prototype et bloc de code)

Question : par quelle fonction commencer le programme ?

Un programme en C :

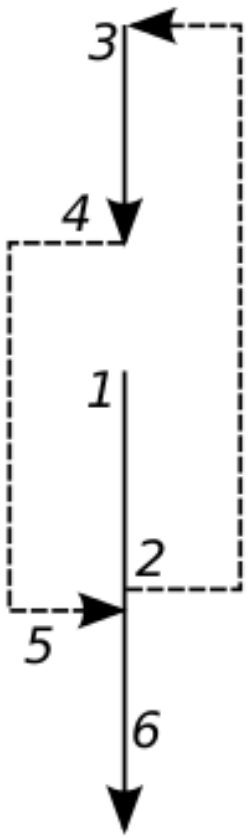
- commence par les commandes pré-processeur (commençant par #)
- enchaîne par une succession de définitions de fonctions (prototype et bloc de code)

Question : par quelle fonction commencer le programme ?

- Une (et une seule) de ces fonctions s'appelle main
- L'exécution du programme commence toujours par la fonction main

• ATTENTION : La lecture de l'algorithme est toujours séquentielle mais pas forcément dans l'ordre de la page ; on appelle cela des sauts.

• À la fin de l'exécution de la fonction (ou procédure), le programme revient juste après à la ligne qui suivait l'appel.



1: Début du programme

2: Appel de la fonction

3: Exécution de la fonction

4: Sortie de la fonction

5: Retour à la ligne suivant l'appel

6: Déroulement normal du programme

Portée d'une variable

Règle d'or : une variable n'est accessible que dans le bloc dans lequel elle a été déclarée.

```c
void count(int min, int max) {
    int var;
    // end DOES NOT EXIST HERE
}

int main() {
    int end;
    end = 10;
    count(0, end);
    // min AND max DOES NOT EXIST HERE
    // var DOES NOT EXIST HERE
    return 0;
}
```

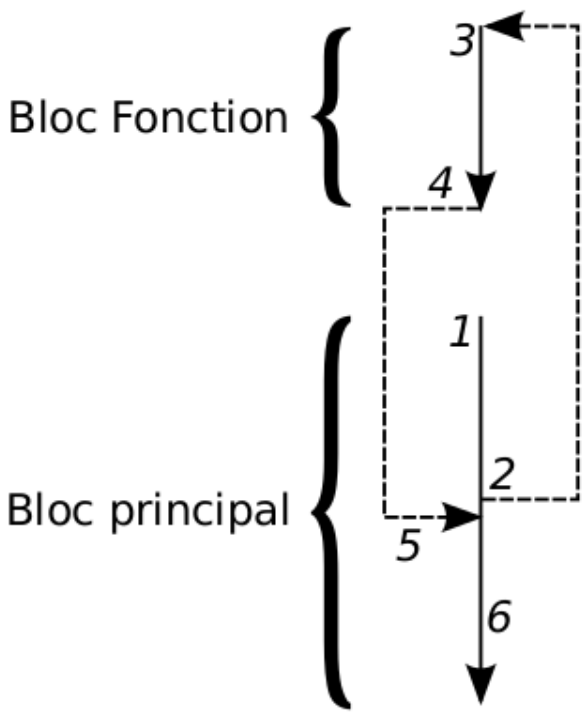
Les paramètres d'entrée

Les paramètres d'entrée (*i.e.* les informations nécessaires à la procédure ou la fonction) sont des variables dont la valeur est déterminée lors de l'appel.

```
...
void count(int min, int max) { // Prototype
    int var;
    for( var = min ; var < max ; var++ ) {
        printf(" %d\n", var);
    }
}
...
int main(){
...
    count(4, 10);
...
}
```

Récupérer le résultat d'une fonction

Une fonction renvoie (avec `return`) une valeur de retour à l'instruction qui l'a appelée.



1: Début du programme

2: Appel de la fonction
(passage des arguments)

3: Exécution de la fonction

4: Sortie de la fonction
(récupération de la valeur de retour)

5: Retour à la ligne suivant l'appel

6: Déroulement normal du programme

Les fonctions permettent de réaliser des calculs et de renvoyer un résultat ou un statut (réussite/échec).

```c
int sum(int var1, int var2) {
    int result;              /* DECLARATION */
    result = var1 + var2; /* CALCUL */
    return result;           /* RETURN */
}
int main() {
    int var;
    var = sum(3, 8);
    // PRINT var
    var = sum(var, 5);
    // PRINT var
    return 0; // MAIN IS A FUNCTION ! IT HAS A RETURN VALUE
}
```

Si une fonction contient plusieurs `return`, le résultat renvoyé est celui obtenu au premier `return` rencontré.

```c
int isEven(int var1) {
    if(var1 % 2 == 0) {
        return 1;
    } else {
        return 0;
    }
}
int main() {
    int var;
    var = 14
    if(isEven(var)==1) {
        // PRINT EVEN
    } else {
        // PRINT ODD
    }
    return 0;
}
```

Exemple de la calculette (tout dans la main):

```c
/* Get the user's choice and check if you got something */
do {
   printf("Please enter the operator (1-5): ");
   if (scanf("%d", &op) != 1) {
      printf("Invalid value!\n");
      return 1;
   }
} while (op < 1 || op > 5);

/* Get the first number and check if you got something */
printf("Enter the first argument: ");
if (scanf("%d", &arg1) != 1) {
   printf("Invalid value!\n");
   return 1;
}

/* Get the second number and check if you got something */
printf("Second argument: ");
if (scanf("%d", &arg2) != 1) {
   printf("Invalid value!\n");
   return 1;
}
```

Exemple de la calculette (avec les fonctions):

```c
#include <stdio.h>
#include <stdlib.h>

int getInteger() {
   int val;
   if (scanf("%d", &val) != 1) {
      printf("Invalid value!\n");
      exit(1);
   }
   return val;
}
```

```c
   /* Get the user's choice and check if you got something */
   do {
      printf("Please enter the operator (1-5): ");
      op = getInteger();
   } while (op < 1 || op > 5);

   /* Get the first number and check if you got something */
   printf("Enter the first argument: ");
   arg1 = getInteger();

   /* Get the second number and check if you got something */
   printf("Second argument: ");
   arg2 = getInteger();
```

Bibliographie

• L. BEAUDOIN, Introduction à l'algorithmique et au langage C (Procédures et fonctions), cours 1A 2016-2017 ESIEA-Paris.

• C. DELANNOY, Langage C, éditions EYROLLES, 4ème tirage 2005.