Chapitre 7 : Les sous-programmes

93

Exemple simple d'appels de sous-programmes

```
#include <stdio.h>
 main(){
  float X, Y;
  printf("Entrez la valeur : ");
  scanf(''%f'',&X);
  Y = cos(X);
  printf(" %f" ,Y);
 }
```

appel du sous-programme d'affichage

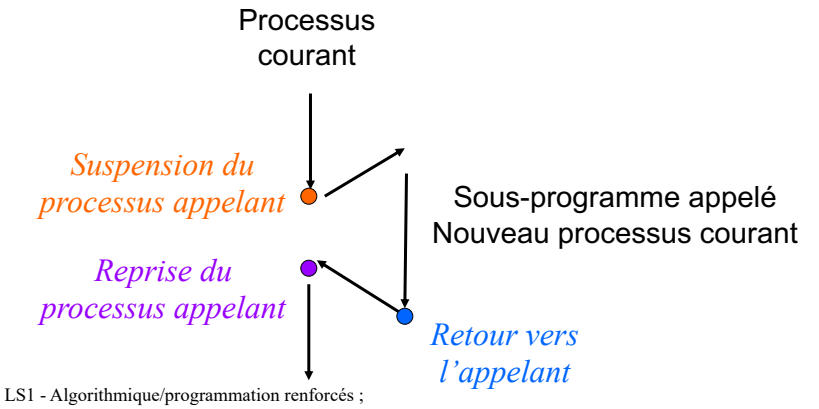
appel du sous-programme de lecture

appel du sous-programme cos

appel du sous-programme d'affichage

94

Fonctionnement



7.2 – Environnement d'un sous-programme

- L'environnement d'un sous-programme est l'ensemble des variables accessibles et des valeurs disponibles dans ce sous-programme, en particulier celles issues du programme appelant.
- Trois sortes de variables sont accessibles dans le sous-programme :
  - Les variables dites globales
  - Les variables dites locales
  - Les paramètres formels

7.2 – Environnement d'un sous-programme

- Les variables dites globales :
  - Celles définies dans le programme appelant et considérées comme disponibles dans le sous-programme
  - Elles peuvent donc être référencées partout dans le programme appelant et dans le sous-programme appelé
  - Leur portée est globale

7.2 – Environnement d'un sous-programme

- Les variables dites locales
  - Celles définie dans le corps du sous-programme, utiles pour son exécution propre et accessibles seulement dans ce sous-programme
  - Elles sont donc invisibles pour le programme appelant
  - Leur portée est locale

7.2 – Environnement d'un sous-programme

- Les paramètres formels
  - Les variables identifiées dans le sous-programme qui servent à l'échange d'information entre les programmes appelant et appelé
  - Leur portée est également locale.

7.3 – Procédures vs Fonctions

- **Définitions**
  - **Procédure** : Un bloc de code qui effectue une tâche spécifique **sans retourner de valeur**.
  - En C, une procédure est une fonction qui retourne `void`.
  - **Fonction** : Un bloc de code qui effectue une tâche spécifique et **retourne une valeur**.

7.3 – Procédures vs Fonctions

- **Exemple de procédure en C**

```c
#include <stdio.h>

// Déclaration et définition d'une procédure
void BonjourProf() {
    printf("Bonjour Monsieur le beau Prof !\n");
}

int main() {
    // Appel de la procédure
    BonjourProf();
    return 0;
}
```


101

7.3 – Procédures vs Fonctions

- **Exemple de fonction en C**

```c
#include <stdio.h>

// Déclaration et définition d'une fonction
int carre(int x) {
    return x * x;
}

int main() {
    int resultat = carre(5); // Appel de la fonction
    printf("Le carré de 5 est %d\n", resultat);
    return 0;
}
```


102

7.4 – Déclaration des procédure et fonction

- **Exemple de fonction en C**

```c
type_de_retour nom_de_la_fonction(type_param1 nom_param1, type_param2 nom_param2, ...) {
    // Corps de la fonction
    return valeur; // Si la fonction retourne une valeur
}
```


7.4 – Appel de Fonctions

- **Passage par valeur**
  - En C, les **paramètres sont passés par valeur** : une copie de la valeur est transmise à la fonction.

```c
#include <stdio.h>

void incrementer(int x) {
    x = x + 1;
    printf("Dans la fonction : %d\n", x);
}

int main() {
    int a = 5;
    incrementer(a);
    printf("Dans main : %d\n", a); // 'a' reste inchangé
    return 0;
}
```

```
Dans la fonction : 6
Dans main : 5
```

7.4 – Appel de Fonctions

- **Passage par référence (pointeurs)**
- En C, pour modifier une variable dans une fonction, on utilise des **pointeurs**.

```c
#include <stdio.h>

void incrementer(int *x) {
    (*x) = (*x) + 1;
}

int main() {
    int a = 5;
    incrementer(&a);
    printf("a = %d\n", a); // 'a' est modifié
    return 0;
}
```

a = 6

105

7.5 – Portée et Durée de Vie des Variables

- **Variables locales**
  - Déclarées à l'intérieur d'une fonction ou d'un bloc.
  - **Portée** : Accessibles uniquement dans le bloc où elles sont déclarées.
  - **Durée de vie** : Existent uniquement pendant l'exécution du bloc.

106

7.5 – Portée et Durée de Vie des Variables

- **Variables locales**

```c
#include <stdio.h>

void maFonction() {
    int x = 10; // Variable locale
    printf("x = %d\n", x);
}

int main() {
    maFonction();
    // printf("%d", x); // Erreur : 'x' n'est pas accessible ici
    return 0;
}
```


107

7.5 – Portée et Durée de Vie des Variables

- **Variables globales**
  - Déclarées en dehors de toute fonction.
  - **Portée** : Accessibles dans tout le programme.
  - **Durée de vie** : Existent pendant toute l'exécution du programme.

108

7.5 – Portée et Durée de Vie des Variables

- **Variables globales**

```c
#include <stdio.h>

int x = 10; // Variable globale

void maFonction() {
    printf("Dans maFonction, x = %d\n", x);
}

int main() {
    printf("Dans main, x = %d\n", x);
    maFonction();
    return 0;
}
```

```
Dans main, x = 10
Dans maFonction, x = 10
```

2025-2026

109

7.5 – Portée et Durée de Vie des Variables

- **Variables statiques**
  - Déclarées avec le mot-clé **`static`**.
  - **Portée** : Locale à la fonction ou au fichier.
  - **Durée de vie** : Existent pendant toute l'exécution du programme.

110

7.5 – Portée et Durée de Vie des Variables

- **Variables statiques**

```c
#include <stdio.h>

void maFonction() {
    static int x = 0; // Variable statique
    x++;
    printf("x = %d\n", x);
}

int main() {
    maFonction(); // x = 1
    maFonction(); // x = 2
    maFonction(); // x = 3
    return 0;
}
```

```
x = 1
x = 2
x = 3
```

2025-2026

111

7.6 – Passage de tableaux à une fonction

- En C, les tableaux sont **passés par référence** (adresse du premier élément).

```c
#include <stdio.h>

void afficherTableau(int tableau[], int taille) {
    for (int i = 0; i < taille; i++) {
        printf("%d ", tableau[i]);
    }
    printf("\n");
}

int main() {
    int monTableau[] = {1, 2, 3, 4, 5};
    afficherTableau(monTableau, 5);
    return 0;
}
```

```
1 2 3 4 5
```

2025-2026

112

7.6 – Passage de tableaux à une fonction

- **Modification d'un tableau dans une fonction**

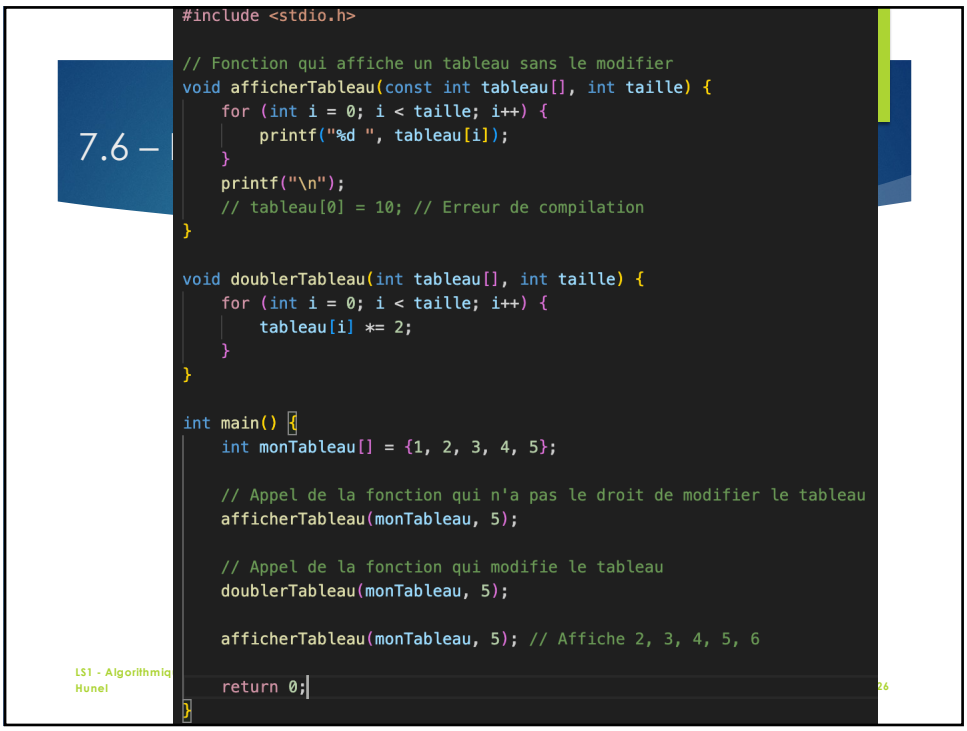
```c
#include <stdio.h>

void doublerTableau(int tableau[], int taille) {
    for (int i = 0; i < taille; i++) {
        tableau[i] *= 2;
    }
}

int main() {
    int monTableau[] = {1, 2, 3, 4, 5};
    doublerTableau(monTableau, 5);
    for (int i = 0; i < 5; i++) {
        printf("%d ", monTableau[i]);
    }
    return 0;
}
```

```
2 4 6 8 10
```

Un tableau étant un pointeur, sa modification à l'intérieur de la fonction entraîne sa modification à l'extérieur

7.6 – Passage de tableaux à une fonction

- Pour garantir la non-modification d'un d'un tableau dans une fonction faire précéder le paramètre tableau de `const`


7.6 –
#include <stdio.h>
// Fonction qui affiche un tableau sans le modifier
void afficherTableau(const int tableau[], int taille) {
for (int i = 0; i < taille; i++) {
printf("%d ", tableau[i]);
}
printf("\n");
// tableau[0] = 10; // Erreur de compilation
}
void doublerTableau(int tableau[], int taille) {
for (int i = 0; i < taille; i++) {
tableau[i] *= 2;
}
}
int main() {
int monTableau[] = {1, 2, 3, 4, 5};
// Appel de la fonction qui n'a pas le droit de modifier le tableau
afficherTableau(monTableau, 5);
// Appel de la fonction qui modifie le tableau
doublerTableau(monTableau, 5);
afficherTableau(monTableau, 5); // Affiche 2, 3, 4, 5, 6
return 0;
}
LS1 - Algorithmiq
Hunel
115