Algorithmique et programmation

Chapitre 6:Fonctions

1. Introduction
2. Déclaration
3. Récursivité

Introduction

• **Programmation structurée** : la décomposition d'un programme complexe en un nombre d'unités (fonctions) plus simple, et donc, facilement compréhensible et maitrisable par le développeur.

• **Fonction**: une unité de programme qui comporte un code exécutant une tâche. Cette tâche peut être exécutée plusieurs fois.

Introduction (suite)

Exemple: un programme utilisant la notion de fonction

```
# include<stdio.h>
float  Pi=3.14;
float  Surface(float r)    /* Déclaration de la fonction*/
{ return (r*r*Pi);}
main()
{
 float  R1=3,R2=5,Res;
Res = Surface(R1);       /* Appel de la fonction pour R1*/
printf ("%f",Res);
Res = Surface(R2);       /* Appel de la fonction pour R2*/
printf ("%f",Res);
}
```

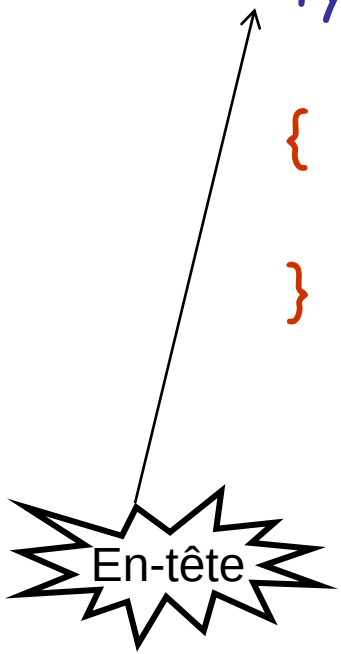
Déclaration

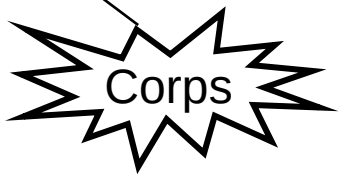
- Une fonction est spécifiée par les caractéristiques suivantes:
  - L'**identificateur** de la fonction
  - Le **type du résultat** retourné par la fonction (spécifié avant l'identificateur)
  - Le **nombre** et le **type** des arguments (les paramètres d'entrée), spécifiés par une liste incluse entre parenthèses et où les éléments sont séparés par des virgules.
  - Le **corps** de la fonction: le code enferme entre les accolades (après la liste des arguments).

Déclaration (suite)

- Une fonction se déclare de la façon suivante:

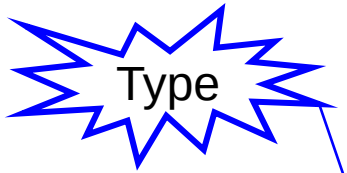
```
type_résultat Identificateur_Fonction (type1
        argument1, type2 argument2, ...)
{
        liste d'instructions
}
```

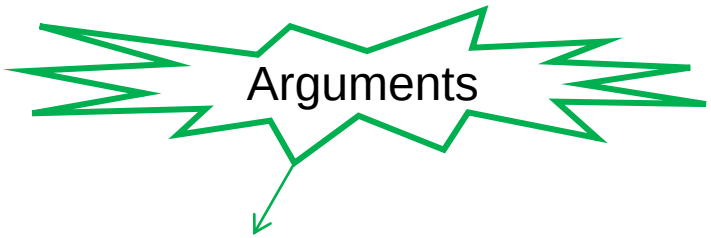




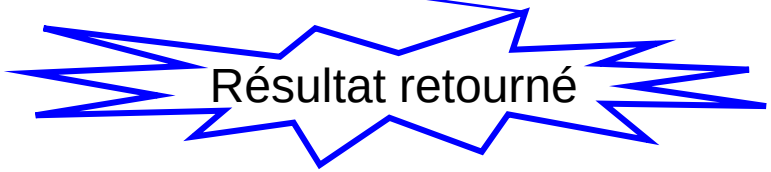
Déclaration (suite)

Exemple: une fonction calculant la surface d'un rectangle






```
float Surface_rectangle (float long, float large)
{
        float res;
        res = long * large;
        return (res);
}
```



Déclaration (suite)

- **Remarques**:
  - Le mot clé **return** permet de sortir de la fonction en retournant une valeur .
  - Si la fonction ne retourne rien, la déclaration de la fonction doit être précédée avec le mot clé **void**.
  - Dans le cas où la fonction n'a pas besoin d'arguments, il faut déclarer la fonction sans paramètres où bien mentionner le mot clé **void** entre les parenthèses.
  - Pour appeler la fonction, il suffit de mettre son nom suivi de la liste des valeurs à donner aux paramètres entre parenthèses.
  - Le **passage** de paramètres s'applique en deux modes: par **valeur** ou par **adresse**.

Déclaration (suite)

Exemple 1: une fonction sans valeur de retour

```
void Surface_rectangle (float long, float large)
{
        float res;
        res = long * larg;
         printf(" la surface est %f \n",res);
}
```

Exemple 2: une fonction sans paramètres et sans valeur de retour

```
void Surface_rectangle (void)
{
        printf(" Exemple de fonction \n");
}
```

Déclaration (suite)

Exemple 3: une fonction sans paramètres et avec valeur de retour

```
float Surface_rectangle (void)
{       float res,long,larg;
        printf("Entrer deux réels:\n");
        scanf("%f,%f",&long,&larg);
        res = long * larg;
        return(res);
}
```

Déclaration (suite)

- **Emplacement** de déclaration:
  - **Avant** la fonction **main()**: la déclaration de la fonction s'effectue sans **prototype**.
  - **Après** la fonction **main()**: la déclaration de la fonction s'effectue avec un **prototype** (mentionné avant **main()**).
- **Prototype**: en-tête de la fonction sans indiquer les noms des arguments.

```
type_résultat Identificateur_Fonction (type1 , type2, ...)
```

Déclaration (suite)

Exemple: Déclaration de la fonction avec un prototype

```
# include<stdio.h>
float  Pi=3.14;
float  Surface(float );    /* Prototype de la fonction*/
main()
{
 float  R1=3,Res;
Res = Surface(R1);        /* Appel de la fonction pour R1*/
printf ("%f",Res);
}
float  Surface(float r)    /* Déclaration de la fonction après main()*/
{ return (r*r*Pi);}
```

Exercice 1

Ecrire un programme en c composé de:

- float **lecture**(): saisir et retourner un réel
- void **solution**(float,float): résoudre une équation du premier degré.
- void **main**()

Exercice 2

Un nombre entier est parfait s'il est égal à la somme de ses diviseurs (sauf lui-même).
Ex : 6 = 1 + 2 + 3 est parfait.

Ecrire une fonction **somme_div** qui retourne la somme des diviseurs d'un nombre passé en paramètre.

Ecrire une fonction **parfait** qui teste si un nombre passé en paramètre est parfait et qui retourne **1** s'il l'est et **0** sinon.

Ecrire un programme principal qui affiche tous les nombres parfaits inférieurs strictement à **1000**.

Exercice 3

Équation de seconde degré

Réaliser un programme en c qui permet de résoudre une équation de seconde degré comportant les fonctions suivantes:

- Saisir un réel: **float lecture()**
- Calculer le discriminant : **float delta(float,float,float)**
- Calculer le nombre de solutions : **int NombreRacines(float)**
- Afficher le nombre de solutions :

void AfficheRacines(float,float,float)

- Calculer les racines de l'équation dans le cas de racines réelles : **float Racine1(float,float,float)**,

float Racine2(float,float,float)

Fonction récursive

- **Récursivité:** une fonction qui comporte un appel à elle-même.

- **Exemple**:

```
int Factoriel (int n)
{
        if(n>1)  return(Factoriel(n-1)*n);
        else return(1);
}
```

Exercice 4

Suite de Fibonacci

Ecrire une fonction en c permettant le calcul de la suite de Fibonacci pour un entier ***n*** en utilisant la récursivité:

$$\begin{cases} U_1 = 1 \\ U_2 = 1 \\ U_n = U_{n-1} + U_{n-2} \quad \textit{pour } n>2 \end{cases}$$

Exercice 4 (solution)

```
int Fibonacci (int n)
{
        if(n==1 || n==2)  return(1);
        else return(Fibonacci(n-1)+Fibonacci(n-2));
}
```

Exercice 5

Puissance:

En utilisant la récursivité, écrire une fonction permettant le calcul de x^n,

Exercice 5 (solution)

```
int Puissance (int n,int x)
{
        if(n==0)  return(1);
        else return(Puissance(n-1,x)*x);
}
```