LES FONCTIONS

Plan

- Introduction aux fonctions
- Méthodes de définition et de déclaration des fonctions
- La portée des variables
- La récursivité

Introduction

- Développer un programme pour résoudre un problème assez complexe nécessite de le découper en plusieurs actions élémentaires afin de faciliter sa conception et sa lisibilité.
- L'intérêt de cette démarche est de diviser un problème en plusieurs sous problèmes moins complexes.
  - *La programmation structurée.*
- Il s'agit de regrouper un ensemble d'instructions faisant partie d'un même objectif au sein d'un même sous programme afin d'y faire appel tant de fois que c'est nécessaire dans les programmes appelants.
  - *Fonction et Procédure.*

Concept de fonction : exemple

Le **nombre de combinaisons** d'une partie à p éléments d'un ensemble à n éléments (avec p ≤ n), noté C_n^p ou $\binom{n}{p}$ (nouvelle notation) que l'on prononce "p parmi n", est le nombre de p-parties différentes d'un ensemble de n objets. L'ordre des objets n'intervient pas. On a :

$$C_n^p = \frac{A_n^p}{p!} = \frac{n!}{p!(n-p)!}$$

ou encore

$$\binom{n}{p} = \frac{n!}{p!(n-p)!}$$

N! est le factoriel de N
N! = 1 * 2 * …*N

Concept de fonction : exemple

```c
#include<stdio.h>
#include<stdlib.h>
//calcul du cnp
main(){
    int n; int p;
    int factn = 1;  int factp = 1;
    int factnp = 1; int cnp = 1;
    int i;
    printf("saisir la valeur de n:  \n");
    scanf("%d", &n);
    printf("saisir la valeur de p:  \n");
    scanf("%d", &p);
    if(n<p)exit(-1);
    //calcul du factoriel de n
    for(i=1; i<=n; i++)
    {
        factn=factn*i;
    }
    //calcul du factoriel de p
    for(i=1; i<=p; i++)
    {
        factp=factp*i;
    }
    //calcul du factoriel de n-p
    for(i=1; i<=n-p; i++)
    {
        factnp=factnp*i;
    }
    cnp = factn/(factp * factnp);
    printf("CNP = %d", cnp);
    system("pause");
}
```

Concept de fonction : exemple

Nom de la fonction

paramètre de la fonction

Type de la valeur retournée

Le mot clé return

```
int fact(int n){
    int factn = 1;
    int i;
    if (n<0) exit(-1);
    else
    for(i=1; i<=n; i++)
    {
        factn=factn*i;
    }
    return factn;
}
//calcul du cnp
```

```
//calcul du cnp
main(){
    int n; int p;
    int factn = 1;  int factp = 1;
    int factnp = 1; int cnp = 1;
    int i;
    printf("saisir la valeur de n:  \n");
    scanf("%d", &n);
    printf("saisir la valeur de p:  \n");
    scanf("%d", &p);
    if(n<p)exit(-1);
    //calcul du factoriel de n
    factn = fact(n);
    //calcul du factoriel de p
    factp = fact(p);
    //calcul du factoriel de n-p
    int np = n-p;
    factnp = fact(np);
    cnp = factn/(factp * factnp);
    printf("CNP = %d", cnp);
    system("pause");
}
```

1er appel de la fonction

Une fonction

- Une fonction est un sous programme dans le but d'effectuer un traitement et de retourner une valeur (résultat du traitement) au programme appelant.
  - *Une fonction est déclarée ainsi :*

```
Type de retour     identifiant_fonction (liste des paramètres)
{
//déclarations des variables locales
//Instructions du traitement
//Le retour de la valeur via le mot clé return
}
```

  - *Exemple : une fonction réalisant la somme de deux entiers*

```
int somme(int a ,  int b)
{
        int s ;
         s= a + b;
        return s;
}
```

Une procédure

- Une procédure est une fonction qui retourne aucune valeur.
- En langage C, une procédure se déclare ainsi

```
void    identifiant_fonction (liste des paramètres)
{
//déclarations des variables locales
//Instructions du traitement
}
```

- Le mot clé **void** est utilisé pour indiquer que la fonction ne retourne pas de valeur.
- Son appel dans le programme appelant s'effectue directement sans assignation à une variable, vu que la procédure ne retourne pas de valeur:

identifiant_fonction (liste des valeurs des paramètres);

Portée des variables

- Dans un programme, on distingue deux types de variables: les variables globales et les variables locales.
- Les variables globales :
  - *Sont déclarées en dehors de tout sous programme;*
  - *Sont accessibles à tous les sous programmes;*
  - *La durée de vie de ces variables est celle de l'algorithme principal.*
- Les variables locales:
  - *Sont déclarées dans la partie de déclaration des fonctions ou des procédures;*
  - *Ne sont accessibles que dans les sous programmes dans lesquels elles sont déclarées;*
  - *Leur durée de vie est celle des sous programmes où elles sont déclarées.*

Méthodes de définition et de déclaration des fonctions

- Le programme (principal (main) ou même une fonction) appelant une fonction doit « connaitre » le **prototype** de cette fonction.
- Plusieurs solutions existent :
- Le programme principal contient la déclaration (**prototype**) de la fonction, le code de la fonction se trouve après le programme principal.
- Le code de la fonction se trouve avant le programme principal.
- La déclaration de la fonction se trouve dans un fichier d'en tête. (**la meilleure solution !**)
- On appelle prototype ou signature d'une fonction sa déclaration de la forme :
  - *type identifiant_fonction (déclaration des pamarètres);*
  - *Exemple : **int somme(int a, int b);***

Méthodes de définition et de déclaration des fonctions

Méthode 1

```c
#include<stdio.h>
#include<stdlib.h>
//la fonction est déclarée et définie avant
//le programme ptincipal
int somme(int a, int b)
{
    int res = a + b;
    return res;
}

main(){
    int a;
    int b;
    int s;
    printf("saisir la valeur de a:  \n");
    scanf("%d", &a);
    printf("saisir la valeur de b:  \n");
    scanf("%d", &b);
    s = somme(a,b);
    printf("la somme des valeur %d et de %d est %d \n", a, b, s);
    system("pause");
}
```

Méthodes de définition et de déclaration des fonctions

Méthode 2:

```c
#include<stdio.h>
#include<stdlib.h>
main(){
    //on déclare le prototype de la fonction somme
    int somme(int a, int b);
    int a;
    int b;
    int s;
    printf("saisir la valeur de a:  \n");
    scanf("%d", &a);
    printf("saisir la valeur de b:  \n");
    scanf("%d", &b);
    s = somme(a,b);
    printf("la somme des valeur %d et de %d est %d \n", a, b, s);
    system("pause");
}
//la fonction est déclarée après le programme ptincipal
int somme(int a, int b)
{
    int res = a + b;
    return res;
}
```

Méthodes de définition et de déclaration des fonctions

Méthode 3

```c
//la fonction est définie
//dans un fichier séparé du programme appelant
//il s agit d'un fichier d'entête ayant l'extension.h
//c'est le fichier "somme.h"
int somme(int a, int b)
{
    int res = a + b;
    return res;
}
```

```c
#include<stdio.h>
#include<stdlib.h>
#include "somme.h"
main(){
    int a;
    int b;
    int s;
    printf("saisir la valeur de a:  \n");
    scanf("%d", &a);
    printf("saisir la valeur de b:  \n");
    scanf("%d", &b);
    s = somme(a,b);
    printf("la somme des valeur %d et de %d est %d \n", a, b, s);
    system("pause");
}
```

Portée des variables

```c
#include<stdio.h>
#include<stdlib.h>
main()
{
    void inc(int x );
    int y = 0;
    printf("Avant appel, valeur de y dans main %d\n", y);
    inc(y);
    printf("Après appel, valeur de y dans main %d\n", y);
    system("pause");
}
void inc(int x )
{
   x = x + 1;
   printf ("Valeur de x dans la fonction %d\n", x);
}
```

Variable globale

Variable locale à la fonction inc

Portée des variables : exemple

```c
int foix2(int n){
    int d;
    d = n*2;
    return d;
}
int divPar2(int a){
    int n;
    n = a/2;
    return n;
}
main(){
    //variable globale
    int n = 5;
    int res = 0;
    //doubler la valeur globale
    res = foix2(n);
    printf("le double de %d est : %d\n", n, res);
    //diviser par deux la valeur globale
    res = divPar2(n);
    printf("le division par 2 de %d est : %d\n", n, res);
    printf("la valeur de la variable globale est : %d\n", n);
    system("pause");
}
```

Variable locale à la fonction foix2

Paramètres des fonctions

Variable locale à la fonction divPar2

Exercice applicatif 1

1. Écrire une fonction **deuxChiffres** qui retourne **vrai** si le nombre passé en paramètre est un nombre positif à 2 chiffres, sinon elle renvoie **faux.**
2. Écrire une fonction **factoriel** qui retourne le factoriel d'un nombre donné;
3. Écrire une procédure **max** qui saisit **n** valeurs et qui affiche la plus grande valeur saisie.
4. Écrire un programme d'essai dans lequel on demande de saisir une valeur **p** positive, puis
   - *L'on indique à l'utilisateur si **p** est un nombre à 2 chiffres*
   - *L'on affiche le factoriel de **p***
   - *L'on demande de saisir **p** valeurs et afficher leur maximum.*

Exercice applicatif 2

Ecrire une procédure qui reçoit en arguments 2 nombres réels et un caractère, et qui affiche un résultat correspondant à l'une des 4 opérations appliquées à ses deux premiers arguments, en fonction de la valeur du dernier, à savoir : addition pour le caractère +, soustraction pour -, multiplication pour * et division pour / (tout autre caractère que l'un des 4 cités sera interprété comme une addition). On tiendra compte des risques de division par zéro.

Récursivité

- On appelle récursive toute fonction ou procédure qui s'appelle elle même.
- La programmation récursive est une technique de programmation qui remplace les instructions de boucle par des appels de fonction.
- Certaines fonctions ne peuvent être définies que récursivement. Plus souvent, une définition récursive est plus élégante, plus simple, plus lisible, plus efficace.
- Exemple :

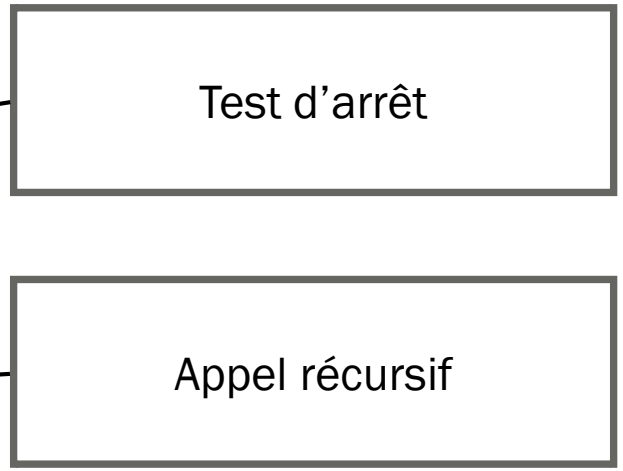
```c
#include<stdio.h>
#include<stdlib.h>
void affiche()
{
    printf("je suis appelée depuis le programme principal\n");
    affiche();

}
main(){
    affiche();
    system("pause");
}
```

- Lorsqu'un algorithme s'appelle lui-même, il est nécessaire que l'enchainement des appels successifs connaisse une fin. La suite des actions à exécuter doit être finie.
- Tout algorithme récursif doit contenir une condition qui assure la fin du nombre d'appels.

Récursivité

```c
#include<stdio.h>
#include<stdlib.h>
void affiche(int n)
{
if(n==0)
    return;
printf("Bonjour \n");
affiche(n-1);
}
main()
{
    affiche(3);
    system("pause");
}
```



Exemple : la fonction du calcul du factoriel

- Le factoriel de N est donné par la formule suivante :
- N!= N * N-1 * N-2 * … * 2 * 1
- Soit : N! = N * (N-1)! avec 0!=1.
- La fonction itérative int **factI**(int n):

```
int factI (int n){
    int i;
    int fact = 1;
    for(i=1; i<=n; i++){
        fact = fact*i;
    }
    return fact;
}
```

- La fonction récursive int **factR**(int n):

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

Récursivité

```
int factR (int n){
    if(n<0) exit(-1);
    else if(n==0)return 1;
    else {
        return n*factR(n-1);
    }
}
main(){
    printf("la fact de 4 est : %d\n", factR(4));
    system("pause");
}
```

```
Appel à fact(4)
.   4*fact(3) = ?
.   Appel à fact(3)
.   .   3*fact(2) = ?
.   .   Appel à fact(2)
.   .   .   2*fact(1) = ?
.   .   .   Appel à fact(1)
.   .   .   .   1*fact(0) = ?
.   .   .   .   Appel à fact(0)
.   .   .   .   Retour de la valeur 1
.   .   .   .   1*1
.   .   .   Retour de la valeur 1
.   .   .   2*1
.   .   Retour de la valeur 2
.   .   3*2
.   Retour de la valeur 6
.   4*6
Retour de la valeur 24
```

Types de récursivité

Récursivité simple : Une fonction simplement récursive est une fonction qui s'appelle elle-même une seule fois.

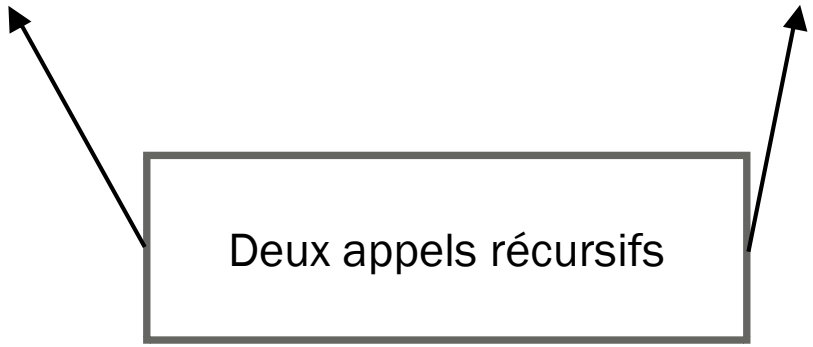
```
int factorielle (int n)
{
   if (n < 0)
      exit (-1);
   else
      if (n == 0)
         return(1);
      else
         return (n * factorielle (n - 1));
}
```

Un seul appel récursif

Types de récursivité

Récursivité multiple : Un programme récursif est multiple si l'un des cas qu'il traite se résout avec plusieurs appels récursifs.

```
int fibonacci(int n)
{
  if ((n==0)||(n==1))
     return (1);
  else
     return (fibonacci(n-1)+fibonacci(n-2));
}
```



Types de récursivité

Récursivité mutuelle : Deux algorithmes sont mutuellement récursifs si l'un fait appel à l'autre et vice-versa.

```
 // n est pair ssi (n-1) est impair
int Pair(int n)
{
  if(n == 0)
     return (1);
  else
     return(Impair(n-1));
}
 // n est impair ssi (n-1) est pair
int Impair(int n)
{
  if(n == 0)
     return (0);
  else
     return(Pair(n-1));
}
```

L'algorithme pair qui appel récursivement l'algorithme impair

L'algorithme impair qui appel récursivement l'algorithme pair

Types de récursivité

Récursivité imbriquée ; Une fonction contient une récursivité imbriquée s'il contient comme paramètre un appel à lui-même. Par exemple, la fonction d'Ackerman $A(m; n)$ définie sur N × N par :

$$A(m,n) = \begin{cases} A(0,n) = n+1 & \text{si } m = 0, \\ A(m,0) = A(m-1,1) & \text{si } n = 0, \\ A(m,n) = A(m-1, A(m,n-1)) & \text{si } m \neq 0 \text{ et } n \neq 0. \end{cases}$$

```
int ack(int m, int n)
{
   if(m == 0)
       return(n + 1);
   else
       if(n == 0)
           return(ack (m - 1, 1));
       else
           return(ack (m - 1,ack (m, n - 1)));
}
```

Appel récursif en paramètre

Types de récursivité

Récursivité terminale : On dit qu'un fonction est récursive terminale, si tout appel récursif est de la forme return f(...).

- Autrement dit, la valeur retournée est directement la valeur obtenue par un appel récursif, sans qu'il n'y ait aucune opération sur cette valeur.

```c
#include<stdio.h>
#include<stdlib.h>
int recTerminal (int n, int a)
{
    if (n==0)   return a;
    else return recTerminal(n-1, n*a);
}
main(){
    int res;
    printf("cette fontion calcul a*n!  \n");
    res=recTerminal(3,4);
    printf("res == %d", res);
    system("pause");
}
```

Une fonction récursive est dite **terminale** si aucun traitement n'est effectué à la remonté d'un appel récursif (sauf le retour d'une valeur).

Types de récursivité

- **Récursivité non terminale:** Une fonction récursive est dite **non terminale** si le résultat de l'appel récursif est utilisé pour réaliser un traitement (en plus du retour d'une valeur).

```
#include<stdio.h>
#include<stdlib.h>
int factT(int n)
{
    int res;
    if (n<0) exit(-1);
    else if( (n==0)||(n==1) )
    {
        res=1;
        return res;
    }
    else
    {
        res= n*factT(n-1);
        printf("res = %d\n", res);
        return res;
    }

}
```

Appel récursif utilisé dans un calcul

Exercices applicatifs 3

Exercice 1:

Programmer le PGCD de deux entiers en utilisant l'algorithme d'Euclide par récursivité.
Principe

si x > y alors PGCD (x, y) = PGCD (y, x)

si 1 ≤ x ≤ y alors PGCD (x, y) = PGCD (x, y modulo x)

si x = 0 alors PGCD (x, y) = y

Exercice 2:

Programmer l'exponentiation binaire de manière récursive. Cette fonction consiste à calculer x^n en appelant $(x*x)^{n/2}$ si n est pair, et $x*x^{n-1}$ si n est impair et retourne 1 si $n = 0$.

Exercice 3:

Ecrire une fonction récursive qui affiche à l'écran des entiers positifs lus au clavier dans l'ordre inverse de leur saisi.
Indication *: Afficher dans l'ordre inverse nécessite une mémorisation des valeurs saisies avant l'affichage. Le fait que la dernière valeur saisie est la première à afficher, l'avant dernière valeur saisie et la deuxième à afficher et ce jusqu'à la première valeur saisie qui sera afficher en dernier pousse à utiliser une pile (la pile de la récursivité).*