TD 6 : Les fonctions

1°) Rendre une valeur

Ecrire la fonction `factoriel(n)` qui prend un `long` en argument et retourne un `long`.

Ecrire la fonction `Cnp(p,n)` qui utilise la fonction `factoriel` pour calculer le nombre de combinaisons de p éléments parmi n. Le résultat sera de type `long`. On rappelle que :

$C^p_n = n ! / (p ! (n-p) !)$

Remarque : ce n'est pas la bonne manière de calculer C^p_n *car les valeurs générées par la fonction factoriel dépassent rapidement la capacité d'un long. Mais l'objectif de cet exercice est seulement d'utiliser des fonctions.*

2°) Paramètres données et paramètres résultats

Ecrire la fonction `evaluerExpression` qui reçoit en argument deux réels et un caractère représentant le code de l'opération a effectuer (+ - * /) et qui rend le résultat de l'évaluation (on supposera dans un premier temps que les données sont toujours valides).

On suppose maintenant que l'expression n'est pas toujours valide. Modifier cette fonction pour que:

- le résultat de la fonction soit un "booléen" qui dit si l'expression est valide ou non,
- un $4^{ème}$ argument soit le résultat du l'évaluation de l'expression si elle est valide.

3°) Tableau en paramètre

Ecrire la fonction `iMini` qui recherche la position du minimum dans un tableau `t` d'entiers entre les indices `a` et `b` (non compris `b`). `t`, `a` et `b` seront passés en paramètre.
Ecrire la fonction `trier` qui utilise cette fonction `iMini` pour trier un tableau de n entiers.

Modifier ces deux fonctions pour se passer de l'argument `a` dans la fonction `iMini`

4°) Un peu d'algorithme

Programmer la fonction réelle `exposant(x,n)` avec x réel et n entier. On tiendra compte des propriétés ci-dessous :

$x^0 = 1$, $x^n = x * x^{n-1}$ si n est impair, $x^n = (x^2)^{n/2}$ si n est pair

Combien faut-il d'itérations pour calculer 2^8 ?
Combien faut-il d'itérations pour calculer 2^{16} ?

Correction TD 6 : Les fonctions

1) Ecrire la fonction `factoriel(n)` qui prend un `long` en argument et retourne un `long`.

Ecrire la fonction `Cnp(p,n)` qui utilise la fonction `factoriel` pour calculer le nombre de combinaison de p éléments parmi n. Le résultat sera de type `long`

```
long factoriel (long n)
{ long f;
  f = 1 ;
  while (n--)
  { f = n*f ;
  }
  return f ;
}

long Cnp(long p, long n)
{ long c ;
  c = factoriel(n)/(factoriel(p)*factoriel(n-p)) ;
}
```

2°)

Ecrire la fonction `evaluerExpression` qui reçoit en argument deux réels et un caractère représentant le code de l'opération a effectuer (+ - * /) et qui rend le résultat de l'évaluation (on supposera dans un premier temps que les données sont toujours valides).

On suppose maintenant que l'expression n'est pas toujours valide. Modifier cette fonction pour que :

- le résultat de la fonction soit un "booléen" qui dit si l'expression est valide ou non,
- un 4ème argument soit le résultat du l'évaluation de l'expression.

```
int evaluerExpression(float a, char op, float b, float *resultat)
{ switch (op)
  { case '+' :
      *resultat = a+b ;
      return 1 ; /* break n'est pas necessaire dans ce cas */
    case '-' :
      *resultat = a-b ;
      return 1 ;
    case '*' :
      *resultat = a*b ;
      return 1 ;
    case '/' :
      if (b !=0)
      { *resultat = a/b ;
        return 1 ;
      }
  }
  return 0 ;
}

void main(void)
```

```
{ float x, y, r;
  char o;
  printf("donnez votre expression\n");
  scanf("%f %c %f", &x, &o, &y);
  if (evaluerExpression(x, o, y, &r))
    printf(" = %d ",r);
  else
    printf("erreur\n");
}
```

3°)
Ecrire la fonction `iMini` qui recherche la position du minimum d'un tableau d'entiers entre les indices `a` et `b` (non compris `b`).
Ecrire la fonction `trier` qui utilise cette fonction `iMini` tour trier un tableau de n entiers.

Modifier ces deux fonctions pour se passer de l'argument `a` dans la fonction `iMini` ;

```
#include <stdio.h>

/*Recherche de la position du plus petit dans une zone [debut, fin[*/
int iMini(int t[], int debut, int fin)
{ int i, min;
  min = debut;
  for (i=debut+1 ; i<fin ; i++)
    if (t[min]>t[i]) min = i;
  return min ;
}

void trier(int t[], int n)
{ int i, m, tmp;
  for (i=0 ; i<n-1 ; i++)
  { m = iMini(t, i, n);
    tmp = t[i];
    t[i] = t[m];
    t[m] = tmp;
  }
}

#define MAX 5
void main(void)
{ int i,tab[MAX]= {3, 2, 5, 10, 1};
  trier(tab, MAX);
  for (i = 0 ; i < MAX ; i++)
    printf("%d ",tab[i]);
}
```

Modifier ces deux fonctions pour se passer de l'argument `a` dans la fonction `iMini`

```
#include <stdio.h>

/* Recherche de la position du plus petit dans un tableau de n elts */
```

```
int iMini(int *t, int n)  /* t pointera sur tab[debut] */
/* int iMini(int t[], int n) écriture équivalente à la précédente */
{ int i, min;
  min = 0;
  for (i=1 ; i<n ; i++)
    if (t[min] > t[i]) min = i;
  return min ;
}

void trier(int tab[], int n)
{ int debut, m, tmp;
  for (debut=0 ; debut<n-1 ; debut++)
  { m = iMini(tab+debut, n-debut);
    tmp = tab[debut];
    tab[debut] = tab[m];
    tab[m] = tmp;
  }
}

#define MAX 5
void main(void)
{ int i, tab[MAX]= {3, 2, 10, 5, 1};

  trier(tab, MAX);
  for (i = 0 ; i < MAX ; i++)
    printf("%d ",tab[i]);
}
```

Programmer la fonction x^n

```
#include <stdio.h>

int pair (int n)
{  return n%2 == 0 ;
}

float exposant(float x, int n)
{  float result;
   result = 1.0;
   while ( n != 0 )
   {  if ( pair(n) )
      {  x = x*x;
         n = n/2;
         /*
      }
      else
      {  result = result * x;
         n = n-1;
      }
   }
```

```
   return result ;
}

float exposantRecursif(float x, int n)
{  float result;
   if (n==0) return 1.0;
   if ( pair(n) ) result = exposantRecursif(x*x, n/2) ;
   else result = x*exposantRecursif(x, n-1);
   return result ;
}

/* variante plus condensée */
float exposantRecursif_2(float x, int n)
{  if (n==0) return 1.0;
   if ( pair(n) ) return exposantRecursif(x*x, n/2) ;
   return x*exposantRecursif(x, n-1);
}

void main(void)
{  float a ;
   int n;
   scanf("%f %d", &a,&n);
   printf("methode non recursive : %f\n", exposant(a,n)) ;
   printf("methode recursive : %f\n", exposantRecursif(a,n)) ;
}
```