LIFAP1 – TP5 : Passage de paramètres

Objectifs : Mode de passage des paramètres : données, données/résultats

1. Kezako (Compréhension : fonction/procédure, paramètres et appels)

   a. Que fait le programme ci-dessous ? Réfléchissez et écrivez le texte que vous pensez voir apparaître à l'exécution de ce programme :

```
#include <iostream>
using namespace std;

void proc_mult(int a, int b, int& ab)
  {
     cout << "execution de la procedure proc_mult" << endl;
     ab = a*b;
}
int fonc_mult(int a, int b)
  {  cout << "execution de la fonction fonc_mult" << endl;
     return a*b;
}
void kezako(int x, int y, int r1, int& r2)
{
     proc_mult( x, y, r1);
    r2 = fonc mult(x,y);
     cout << "A la fin de kezako r1=" << r1 << " r2=" << r2 <<endl;
}

int main(void)
{  int a, y, res1, res2;
   a = 5;        y = 4;       res1 = 0;        res2 = 1;
     cout << "Dans main avant kezako res1=" << res1 << " res2=" <<
res2  <<endl;
       kezako( a, y, res1, res2);
       cout << "Dans main apres kezako res1=" << res1 << " res2="<<
res2 << endl;
     return 0;
}
```

   b. Téléchargez le programme kezako à partir de la page du cours afin de vérifier si votre intuition est correcte. Sinon demandez une explication à votre encadrant de TP.

```
        #include <iostream>
        Using namespace std;
        void proc_mult(int a, int b, int& ab)
        {
5       cout << "execution de la procedure proc_mult" << endl;
6       ab = a*b;
        }
        int fonc_mult(int a, int b)
8       {  cout << "execution de la fonction fonc_mult" << endl;
9       return a*b;
        }
        void kezako(int x, int y, int r1, int& r2)
        {
4       proc_mult( x, y, r1);
7 9       r2 = fonc_mult(x,y);
10      cout << "A la fin de kezako r1=" << r1 << " r2=" << r2 <<endl;
        }
1       int main(void)
        {  int a, y, res1, res2;
           a = 5;       y = 4;      res1 = 0;       res2 = 1;
2       cout << "Dans main avant kezako res1=" << res1 << " res2=" << res2  <<endl;
```

```
3       kezako( a, y, res1, res2);
11        cout << "Dans main apres kezako res1=" << res1 << " res2="<< res2 << endl;
12      return 0;
}
```

Exécution du programme :

1 Le programme commence toujours par la fonction `main`. Les variables sont crées et prennent leur valeur initiale. `a=5, y=4, res1=0, res2=1`

2 Le programme affiche la première ligne :
Dans main avant kezako res1=0 res2=1

3 Appel à la fonction `kezako` en passant `a` pour `x(x=5),y` pour `y(y=4),res1` pour `r1(r1=0)` et `res2` pour `r2(`***`r2=res2=1`***`)`. Le passage par référence se fait entre `res2` et `r2`.

4 Appel à la fonction `proc_mult` en passant `x` pour `a(a=5), y` pour `b(b=4),r1` pour `ab (`***`ab=r1=0`***`)`. Le passage par référence se fait entre r1 et ab.

5 Dans proc_mult, une ligne est affichée :
execution de la procedure proc_mult

6 Dans proc_mult, ab prend pour valeur a*b (5*4=20), ab=20. A la fin de proc_mult, ab=20 donc r1=20.

7 Dans kezako, appel à la fonction fonc_mult en passant x pour a (a=5) et y pour b(b=4)

8 Dans fonc_mult, une ligne est affichée :
execution de la fonction fonc_mult

9 fonc_mult renvoie a*b=5*4=20. Dans kezako, r2 prend pour valeur 20.

10 Dans kezako, une ligne est affiché :
A la fin de kezako r1= 20 r2= 20
A la fin de kezako, r2=20, donc res2 = 20

11 Dans main, la dernière ligne est affichée :
Dans main apres kezako res1=0 res2=20
res1 reste inchangé mais res2 a comme valeur 20.

12 Fin de programme.

2. Passage de paramètres. Ecrire pour chaque exercice le sous-programme demandé ainsi que le programme principal permettant de le tester.
   a. Permutation circulaire de 3 variables

```
void permutationcirculaire(int &a, int &b, int &c)
{
  int tampon;
  tampon=c;
  c=b;
  b=a;
  a=tampon;
}
int main (void)
{
  int v1,v2,v3;
  cout<<"donnez la premiere valeur";
  cin>>v1;
  cout<<endl;
  cout<<"donnez la deuxieme valeur";
  cin>>v2;
  cout<<endl;
  cout<<"donnez la troisieme valeur";
  cin>>v3;
  cout<<endl;
  permutationcirculaire(v1,v2,v3);
  cout<<"apres permutation : nouvelles valeurs"<<v1<<"  "<<v2<<"  "<<v3;
  return 0;
}
```

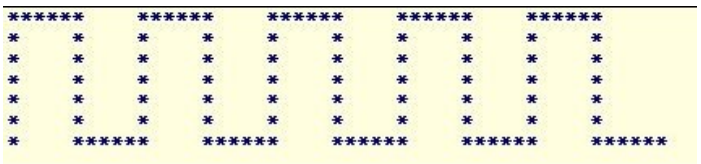
b. Nombre de combinaisons (fonction **ET** procédure)

```
int factorielle (int n)
{
  int fact=1;
  for(int i=2 ; i<=n ; i++)
    fact*=i;
  return fact;
}
int combinaisonF(int n, int p)
{
  return factorielle(n)/(factorielle(p)*factorielle(n-p));
}

void combinaisonP(int n, int p, int &combP)
{
  combP=factorielle(n)/(factorielle(p)*factorielle(n-p));
}

int main (void)
{
  int n,p,cp;

  cout<<"donnez la  valeur de n";
  cin>>n;
  cout<<endl;
  cout<<"donnez la valeur de p";
  cin>>p;
  cout<<endl;
  cout<<factorielle(n);
  cout<<combinaisonF(n,p);
  combinaisonP(n,p,cp);
  cout<<cp;
  return 0;

}
```

3. Écrivez une procédure qui affiche une frise.

```
procedure afficherFrise(n,l,h : donnée Entier)
// n est le nombre de fois que se repete le motif
// l est la demi-longueur d'un motif
// h est la hauteur d'un motif
```

Exemple :
afficherFrise(5, 6, 7);



```
void aff_frise(int n, int l, int h)
{
  int i,j;
  char e=' ';
  //~ aff_ncar(1,'*');
  for(i=0;i<n;i++)
  {
    aff_ncar(l,'*');
    aff_ncar(l-2,e);
  }
  cout<<endl;
```

```
  for(i=1;i<h-1;i++)
  {
    for(j=0;j<n;j++)
    {
      aff_ncar(1,'*');
      aff_ncar(l-2,e);
      aff_ncar(1,'*');
      aff_ncar(l-2,e);
    }
    cout<<endl;
  }

  aff_ncar(1,'*');
  for(i=0;i<n;i++)
  {
    aff_ncar(l-2,e);
    aff_ncar(l,'*');
  }
  cout<<endl;

}
```