LIFAP1 – TP3 : Passage de paramètres / GRAPIC

Objectifs : Mode de passage des paramètres : données, données/résultats
Prise en main de GRAPIC

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. **Prise en main de l'outil GRaPiC : Graphics for Algo/Prog in C/C++**
   Pour réaliser les exercices demandés, reprenez vos notes du TD 5 et n'hésitez pas à consulter les tutoriels disponibles en ligne :
   http://liris.cnrs.fr/alexandre.meyer/grapic/html/index.html .

   a. Téléchargez la dernière version de GRaPiC et installez-la sur votre compte. Ouvrez ensuite le fichier `Grapic/build/windows/grapic_workspace` avec `Codeblocks`. Sélectionnez dans la partie gauche de la fenêtre le projet que vous souhaitez utiliser (`LIFAP1_TP3`) puis à l'aide du bouton droit, choisissez `Activate project`.

Vous trouverez une copie des résultats qu'on souhaite obtenir dans les questions suivantes à la fin de l'énoncé.

b. Ecrivez un sous-programme qui affiche au centre de la fenêtre un carré dont la taille sera passée en paramètre. Vous utiliserez la procédure GRaPiC `Rectangle`.

```
void draw_carre(int taille)
{
   int i;
   rectangle( DIMW/2 - taille/2, DIMW/2 - taille/2, DIMW/2 + taille/2, DIMW/2 + taille/2);

}
```

c. Réutilisez la procédure précédente pour afficher n (passé en paramètres) carrés imbriqués.

```
void draw_carre_imbrique(int n)
{
   int i;
   for(i=0;i<n;i++)
   {
      draw_carre(10*i);
   }
}
```

d. Ecrivez un sous-programme qui dessine un cercle plein dont le rayon et la couleur sont passés en paramètres. Vous utiliserez pour cela `CircleFill`.

```
void draw_cercle(int rayon, int couleur_R, int couleur_G, int couleur_B)
{
   int i;

   color(couleur_R,couleur_G, couleur_B);
   circleFill(  DIMW/2, DIMW/2, rayon);
}
```

e. En utilisant la procédure précédente, dessinez à présent une cible (alternance de cercles concentriques noirs et blancs).

```
void draw_cible(int taille)
{
   int i, couleur;
   for(i=0;i<taille;i++)
   {
      if (i%2 == 0)
         couleur=0;
      else
         couleur = 255;
      color(couleur,couleur, couleur);
      circleFill(DIMW/2, DIMW/2, DIMW/2 - i*DIMW/2/taille);
   }
}
```

f. Dessinez une grille de carrés produisant un dégradé de couleur du vert (dans le coin supérieur gauche) au rouge (dans le coin inférieur droit).

```
void draw_degrade()
{
   int i,j;
   const int l = DIMW/10;
   for(i=0;i<10;++i)
      for(j=0;j<10;++j)
      {
         color( 255*i*10/100, 255*j*10/100, 0);
         rectangleFill( i*l+1, j*l+1, (i+1)*l-1, (j+1)*l-1 );
      }

}
```

g. Ecrivez un sous-programme qui affiche une image dont la taille diminue au cours du temps.

```
void draw_retrecit(Image im)
{
   int i;
   const int nb_frame = 30;
   for (i=0;i<nb_frame;i++)
   {
      winClear();
      image_draw( im, DIMW/2-(400-i*400/nb_frame)/2, DIMW/2-(400-
i*400/nb_frame)/2, 400-i*400/nb_frame, 400-i*400/nb_frame);
      winDisplay();
      delay(2000/nb_frame);
   }
}
```

h. En utilisant 3 images différentes, générez une grille d'images positionnées aléatoirement.

```
void draw_grille_image(Image im1,Image im2,Image im3)
{
 int i,j,num_image;
 const int nb_line = 6;
 const int nb_row = 6;
   for(i=0;i<nb_line;i++)
      for(j=0;j<nb_row;j++)
      {
         num_image = rand () % 3;
         if (num_image ==0) image_draw( im1, i*(DIMW/nb_line), j*(DIMW/nb_row),
(DIMW/nb_line), (DIMW/nb_row));
         else if (num_image ==1) image_draw(im2, i*(DIMW/nb_line), j*(DIMW/nb_row),
(DIMW/nb_line), (DIMW/nb_row));
         else image_draw(im3, i*(DIMW/nb_line), j*(DIMW/nb_row), (DIMW/nb_line),
(DIMW/nb_row));
      }
   delay(5000);
}
```

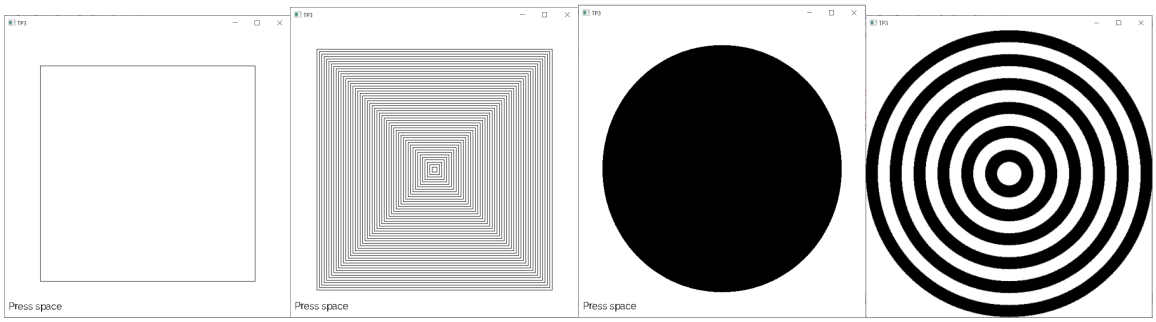
i. Sans utilisez la procédure Circle de GRaPiC, dessinez un cercle plein dont le rayon `r` est passé en paramètre et contenant le caractère `c` également passé en paramètre.
En supposant que le centre du cercle est au centre de la fenêtre, vous utiliserez le calcul de la distance au centre de tout point de la fenêtre pour déterminer s'il appartient ou non au cercle.

Rappel : Soient A (x, y) et O (x_0, y_0) alors la distance entre A et O est $\sqrt{(x-x_0)^2+(y-y_0)^2}$

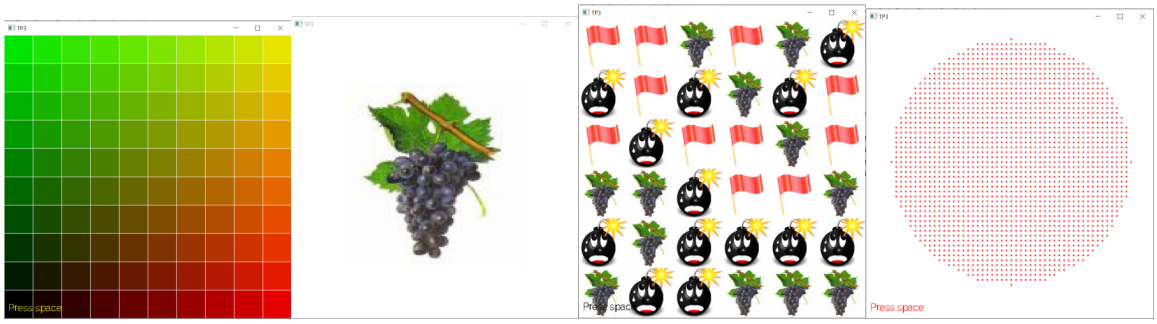
```
void draw_cercle_caractere(int r, char c[1])
{
   int i,j;
   float abscisse, ordonnee,dist;

    for(i=0;i<DIMW;i+=10)
    {
       for(j=0;j<DIMW;j+=10)
      {
         dist = sqrt(pow(i-DIMW/2,2)+pow(j-DIMW/2,2));
         if (dist<=r)
         {
            color(255,0,0);
            print(i,j,c);
         }
```

```
        }
    }
}
```


Press space
Press space
Press space

b *c* *d* *e*


Press space
Press space

f *g* *h* *i*