

Solutions des exercices du Cours 6

Les fonctions

Qu'affichent ces programmes ?

A:

```
float f(float m, float p)
{
    float y = m / p;
    return y * y;
}
```

...

```
cout << f(16, 2) << endl;
float m = f(2, 1), p = f(8, 2), y = f(15, 3);
cout << m << " " << p << " " << y << endl;
y = f(p, m);
cout << y << endl;
```

B:

```
float a(float m, float p)
{
    return m + p;
}
```

```
float b(float m, float p)
{
    return m * p;
}
```

```
float c(float m)
{
    return b(m, m);
}
```

...

```
cout << a(5, 2) << " " << b(5, 2) << endl;
cout << a( b(3, 2), a(5, 4) ) << endl;
cout << c(5) << endl;
```

A:

```
float f(float m, float p)
{
    float y = m / p;
    return y * y;
}
```

?

...

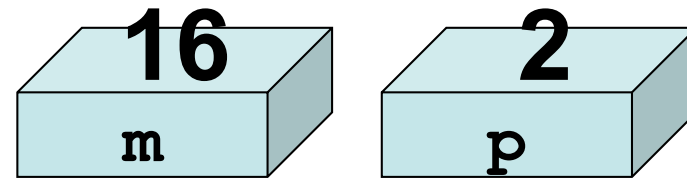
```
→ cout << f(16, 2) << endl;
float m = f(2, 1), p = f(8, 2), y = f(15, 3);
cout << m << " " << p << " " << y << endl;
y = f(p, m);
cout << y << endl;
```

A:

```
→ float f(float m, float p)
{
    float y = m / p;
    return y * y;
}
```

?

```
→ ...
cout << f(16, 2) << endl;
float m = f(2, 1), p = f(8, 2), y = f(15, 3);
cout << m << " " << p << " " << y << endl;
y = f(p, m);
cout << y << endl;
```

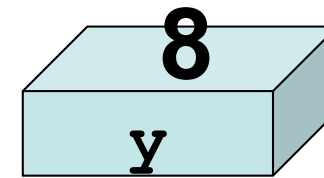
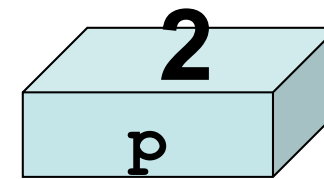
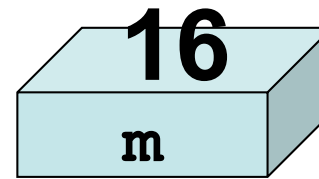


A:

```
→ float f(float m, float p)
{
    float y = m / p;
    return y * y;
}
```

?

```
→ ...
cout << f(16, 2) << endl;
float m = f(2, 1), p = f(8, 2), y = f(15, 3);
cout << m << " " << p << " " << y << endl;
y = f(p, m);
cout << y << endl;
```



A:

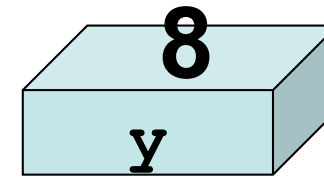
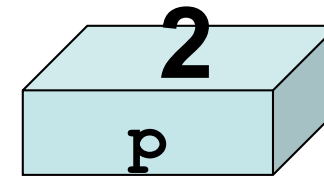
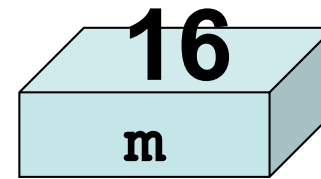
```
float f(float m, float p)
{
    float y = m / p;
    return y * y;
}
```



...



```
cout << f(16, 2) << endl;
float m = f(2, 1), p = f(8, 2), y = f(15, 3);
cout << m << " " << p << " " << y << endl;
y = f(p, m);
cout << y << endl;
```



A:

```
float f(float m, float p)
{
    float y = m / p;
    return y * y;
}
```

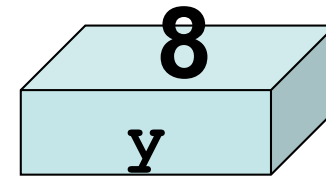
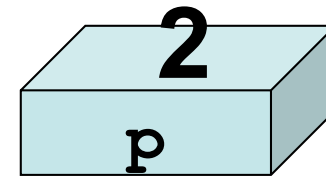
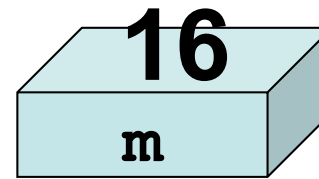
8*8=64



...



```
cout << f(16, 2) << endl;
float m = f(2, 1), p = f(8, 2), y = f(15, 3);
cout << m << " " << p << " " << y << endl;
y = f(p, m);
cout << y << endl;
```



A:

```
float f(float m, float p)
{
    float y = m / p;
    return y * y;
}
```

```
...
→ cout << f(16, 2) << endl;
float m = f(2, 1), p = f(8, 2), y = f(15, 3);
cout << m << " " << p << " " << y << endl;
y = f(p, m);
cout << y << endl;
```

A:

```
float f(float m, float p)
{
    float y = m / p;
    return y * y;
}
```

```
...
→ cout << f(64, 2) << endl;
float m = f(2, 1), p = f(8, 2), y = f(15, 3);
cout << m << " " << p << " " << y << endl;
y = f(p, m);
cout << y << endl;
```

64

A:

```
float f(float m, float p)
{
    float y = m / p;
    return y * y;
}
```

...

```
cout << f(16, 2) << endl;
```

```
→ float m = f(2, 1), p = f(8, 2), y = f(15, 3);
cout << m << " " << p << " " << y << endl;
y = f(p, m);
cout << y << endl;
```

64

A:

```
float f(float m, float p)
{
    float y = m / p;
    return y * y;
}
```

La fonction f renvoie le carré du rapport du premier paramètre et du deuxième paramètre.

...

```
cout << f(16, 2) << endl;
```

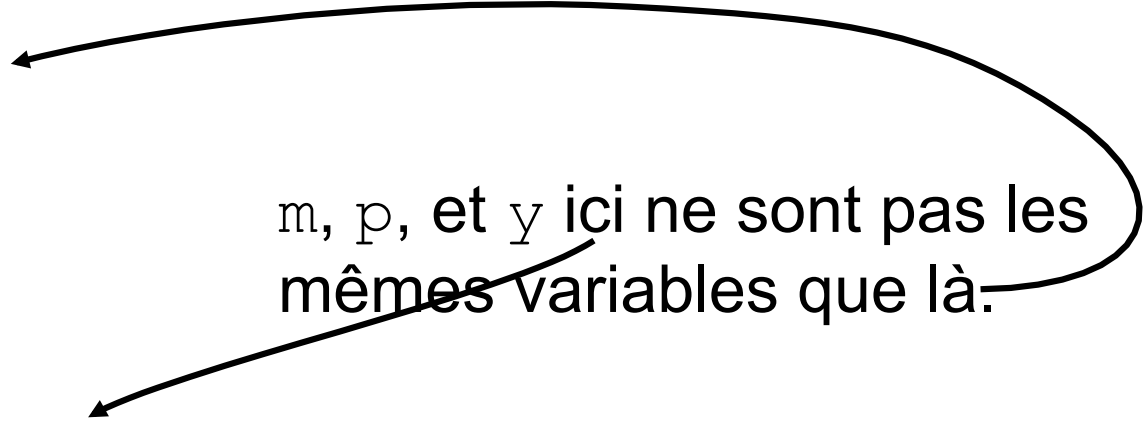
```
→ float m = f(2, 1), p = f(8, 2), y = f(15, 3);
cout << m << " " << p << " " << y << endl;
y = f(p, m);
cout << y << endl;
```

64

A:

```
float f(float m, float p)
{
    float y = m / p;
    return y * y;
}
```

m, p, et y ici ne sont pas les
mêmes variables que là.



...

```
cout << f(16, 2) << endl;
```

```
→ float m = f(2, 1), p = f(8, 2), y = f(15, 3);
```

```
cout << m << " " << p << " " << y << endl;
```

```
y = f(p, m);
```

```
cout << y << endl;
```

64

A:

```
float f(float m, float p)
{
    float y = m / p;
    return y * y;
}
```

La fonction f renvoie le carré du rapport du premier paramètre et du deuxième paramètre.

...

```
cout << f(16, 2) << endl;
```

```
→ float m = f(2, 1), p = f(8, 2), y = f(15, 3);
cout << m << " " << p << " " << y << endl;
y = f(p, m);
cout << y << endl;
```

64

4
m

16
p

25
y

A:

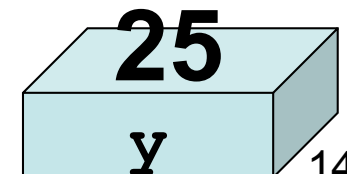
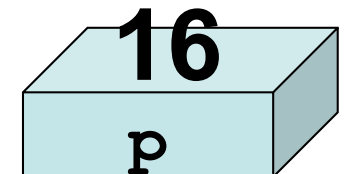
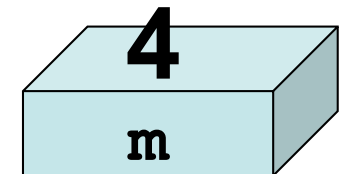
```
float f(float m, float p)
{
    float y = m / p;
    return y * y;
}
```

La fonction f renvoie le carré du rapport du premier paramètre et du deuxième paramètre.

...

```
cout << f(16, 2) << endl;
float m = f(2, 1), p = f(8, 2), y = f(15, 3);
→ cout << m << " " << p << " " << y << endl;
y = f(p, m);
cout << y << endl;
```

```
64
4 16 25
```



A:

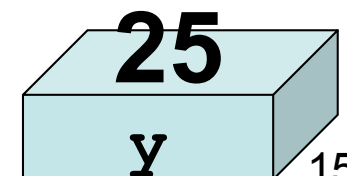
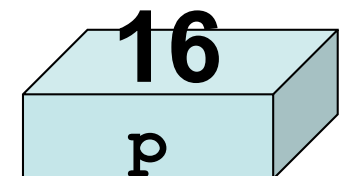
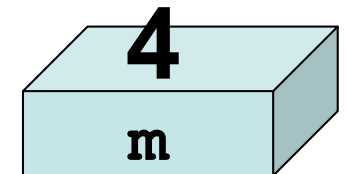
```
float f(float m, float p)
{
    float y = m / p;
    return y * y;
}
```

La fonction f renvoie le carré du rapport du premier paramètre et du deuxième paramètre.

...

```
cout << f(16, 2) << endl;
float m = f(2, 1), p = f(8, 2), y = f(15, 3);
cout << m << " " << p << " " << y << endl;
→ y = f(p, m);
cout << y << endl;
```

```
64
4 16 25
```



A:

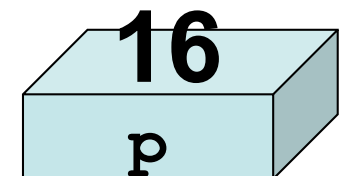
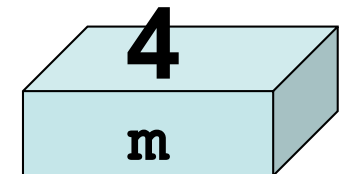
```
float f(float m, float p)
{
    float y = m / p;
    return y * y;
}
```

La fonction f renvoie le carré du rapport du premier paramètre et du deuxième paramètre.

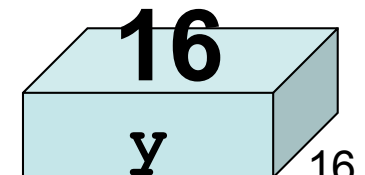
...

```
cout << f(16, 2) << endl;
float m = f(2, 1), p = f(8, 2), y = f(15, 3);
cout << m << " " << p << " " << y << endl;
→ y = f(p, m);
cout << y << endl;
```

```
64
4 16 25
```



$$(16/4)*(16/4)=$$



A:

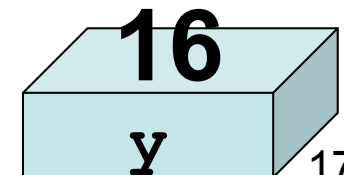
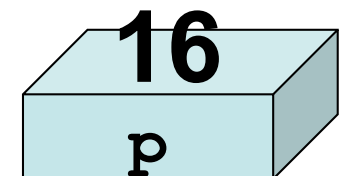
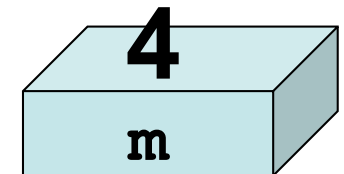
```
float f(float m, float p)
{
    float y = m / p;
    return y * y;
}
```

La fonction f renvoie le carré du rapport du premier paramètre et du deuxième paramètre.

...

```
cout << f(16, 2) << endl;
float m = f(2, 1), p = f(8, 2), y = f(15, 3);
cout << m << " " << p << " " << y << endl;
y = f(p, m);
→ cout << y << endl;
```

```
64
4 16 25
16
```



Pour trouver ce que fait la ligne

```
y = f(p, m);
```

il suffit

- de voir que la fonction `f` renvoie le carré du rapport du premier paramètre et du deuxième paramètre, et
- de se rappeler que `y`, `p`, et `m` dans cette ligne font référence:
 - aux variables déclarées dans la fonction `main` et initialisées à la ligne

```
float m = f(2, 1), p = f(8, 2), y = f(15, 3);
```

- et PAS aux paramètres `m`, `p` et à la variable locale `y` de la fonction `f`

```
float f(float m, float p)
{
    float y = m / p;
```

Les transparents suivants détaillent néanmoins ce qui se passe quand la ligne

```
y = f(p, m);
```

est exécutée, même si en pratique, il n'y a pas à considérer un tel niveau de détail pour trouver la valeur de `y`, juste ce que fait la fonction `f`.

A:

```
float f(float m, float p)
{
    float y = m / p;
    return y * y;
}
```

...

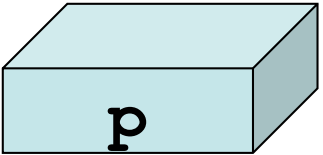
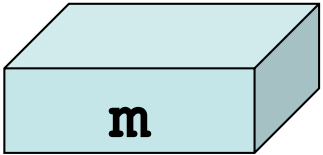
```
cout << f(16, 2) << endl;
float m = f(2, 1), p = f(8, 2), y = f(15, 3);
cout << m << " " << p << " " << y << endl;
→ y = f(p, m);
cout << y << endl;
```

```
64
4 16 25
```

4
m

16
p

25
y



correspond à

A:

```

float f(float m, float p)
{
    float y = m / p;
    return y * y;
}

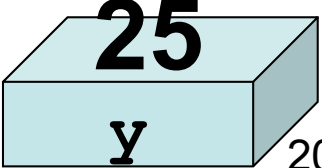
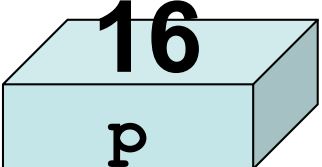
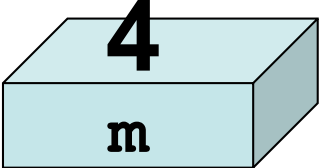
```

...

```

cout << f(16, 2) << endl;
float m = f(2, 1), p = f(8, 2), y = f(15, 3);
cout << m << " " << p << " " << y << endl;
y = f(p, m);
cout << y << endl;

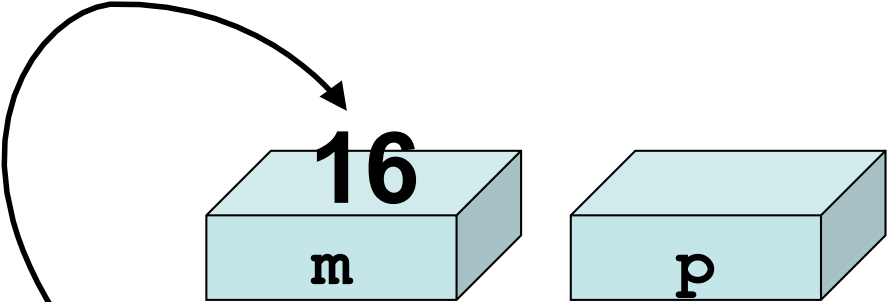
```



```

64
4 16 25

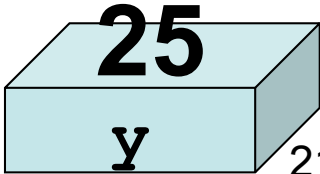
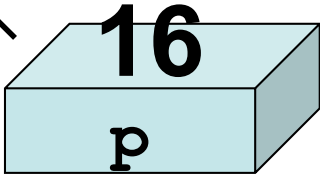
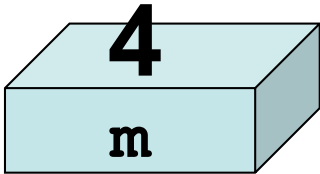
```



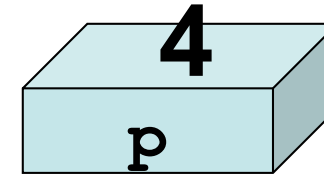
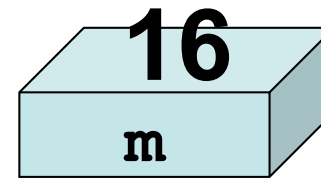
La valeur de la variable `p` est copiée dans le paramètre `m`

```
A:  
→ float f(float m, float p)  
{  
    float y = m / p;  
    return y * y;  
}
```

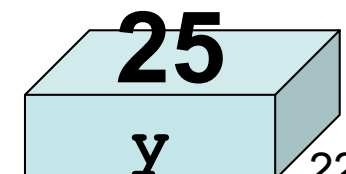
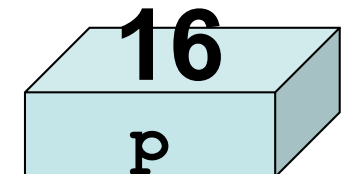
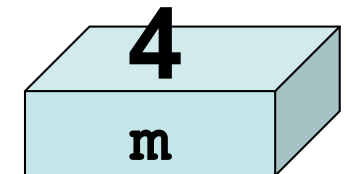
```
...  
cout << f(16, 2) << endl;  
float m = f(2, 1), p = f(8, 2), y = f(15, 3);  
cout << m << " " << p << " " << y << endl;  
→ y = f(p, m);  
cout << y << endl;
```



```
64  
4 16 25
```



même chose pour la variable
m dans le paramètre p



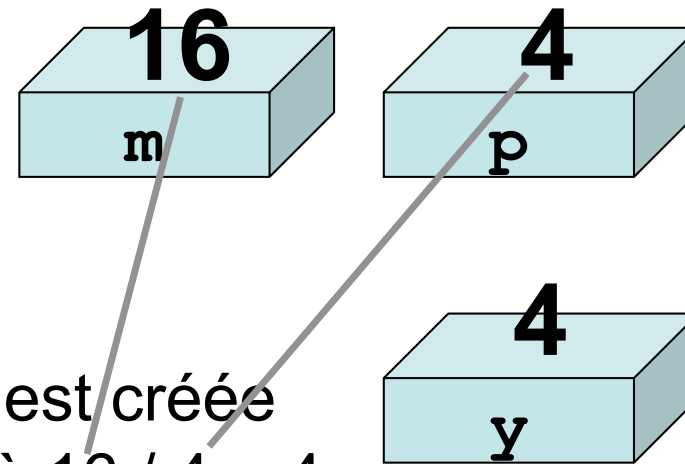
A:

```
→ float f(float m, float p)
{
    float y = m / p;
    return y * y;
}
```

...

```
cout << f(16, 2) << endl;
float m = f(2, 1), p = f(8, 2), y = f(15, 3);
cout << m << " " << p << " " << y << endl;
→ y = f(p, m);
cout << y << endl;
```

```
64
4 16 25
```



A:

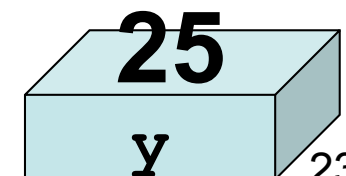
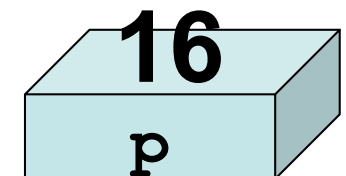
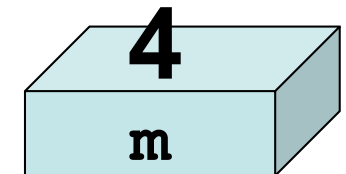
```
float f(float m, float p)
{
  float y = m / p;
  return y * y;
}
```



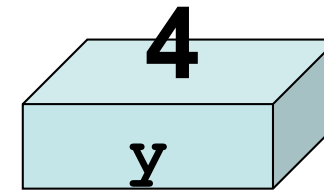
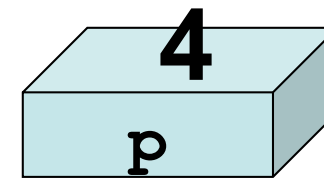
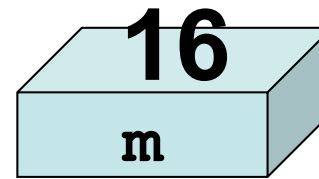
la variable y est créée
et initialisée à $16 / 4 = 4$

...

```
cout << f(16, 2) << endl;
float m = f(2, 1), p = f(8, 2), y = f(15, 3);
cout << m << " " << p << " " << y << endl;
y = f(p, m);
cout << y << endl;
```



```
64
4 16 25
```



A:

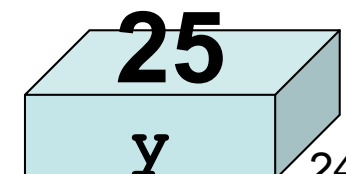
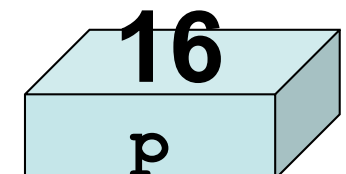
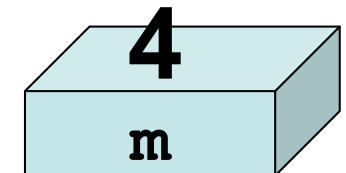
```
float f(float m, float p)
{
    float y = m / p;
    return y * y;
}
```

la fonction retourne la valeur $4 * 4 = 16$



...

```
cout << f(16, 2) << endl;
float m = f(2, 1), p = f(8, 2), y = f(15, 3);
cout << m << " " << p << " " << y << endl;
y = f(p, m);
cout << y << endl;
```



```
64
4 16 25
```

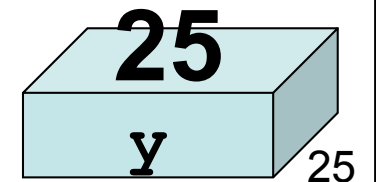
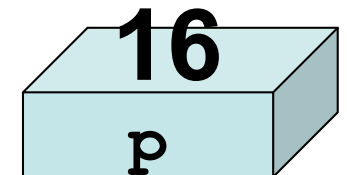
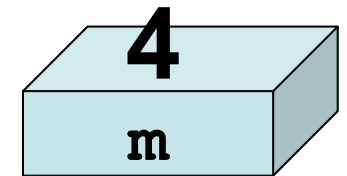
A:

```
float f(float m, float p)
{
    float y = m / p;
    return y * y;
}
```

...

```
cout << f(16, 2) << endl;
float m = f(2, 1), p = f(8, 2), y = f(15, 3);
cout << m << " " << p << " " << y << endl;
→ y = f(16, m);
cout << y << endl;
```

```
64
4 16 25
```



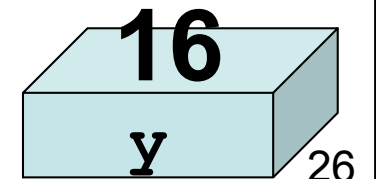
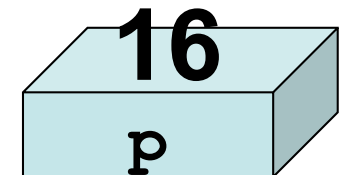
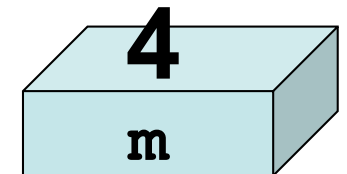
A:

```
float f(float m, float p)
{
    float y = m / p;
    return y * y;
}
```

...

```
cout << f(16, 2) << endl;
float m = f(2, 1), p = f(8, 2), y = f(15, 3);
cout << m << " " << p << " " << y << endl;
→ y = f(16, m);
cout << y << endl;
```

```
64
4 16 25
```



B:

```
float a(float m, float p)
{
    return m + p;
}
```

```
float b(float m, float p)
{
    return m * p;
}
```

```
float c(float m)
{
    return b(m, m);
}
```

...

```
cout << a(5, 2) << " " << b(5, 2) << endl;
cout << a( b(3, 2), a(5, 4) ) << endl;
cout << c(5) << endl;
```

B:

```
float a(float m, float p)
{
    return m + p;
}
```

```
float b(float m, float p)
{
    return m * p;
}
```

```
float c(float m)
{
    return b(m, m);
}
```

...

```
cout << a(5, 2) << " " << b(5, 2) << endl;
cout << a( b(3, 2), a(5, 4) ) << endl;
cout << c(5) << endl;
```

**La fonction a renvoie la
somme des paramètres.**

B:

```
float a(float m, float p)
{
    return m + p;
}
```

La fonction `a` renvoie la somme des paramètres.

```
float b(float m, float p)
{
    return m * p;
}
```

La fonction `b` renvoie le produit des paramètres.

```
float c(float m)
{
    return b(m, m);
}
```

...

```
cout << a(5, 2) << " " << b(5, 2) << endl;
cout << a( b(3, 2), a(5, 4) ) << endl;
cout << c(5) << endl;
```

B:

```
float a(float m, float p)
{
    return m + p;
}
```

La fonction `a` renvoie la somme des paramètres.

```
float b(float m, float p)
{
    return m * p;
}
```

La fonction `b` renvoie le produit des paramètres.

```
float c(float m)
{
    return b(m, m);
}
```

La fonction `c` renvoie le carré du paramètre.

...

```
cout << a(5, 2) << " " << b(5, 2) << endl;
cout << a( b(3, 2), a(5, 4) ) << endl;
cout << c(5) << endl;
```

B:

```
float a(float m, float p)
{
    return m + p;
}
```

La fonction `a` renvoie la somme des paramètres.

```
float b(float m, float p)
{
    return m * p;
}
```

La fonction `b` renvoie le produit des paramètres.

```
float c(float m)
{
    return b(m, m);
}
```

La fonction `c` renvoie le carré du paramètre.

...

```
→ cout << a(5, 2) << " " << b(5, 2) << endl;
   cout << a( b(3, 2), a(5, 4) ) << endl;
   cout << c(5) << endl;
```

B:

```
float a(float m, float p)
{
    return m + p;
}
```

La fonction `a` renvoie la somme des paramètres.

```
float b(float m, float p)
{
    return m * p;
}
```

La fonction `b` renvoie le produit des paramètres.

```
float c(float m)
{
    return b(m, m);
}
```

La fonction `c` renvoie le carré du paramètre.

...

```
→ cout << a(5, 2) << " " << b(5, 2) << endl;
   cout << a( b(3, 2), a(5, 4) ) << endl;
   cout << c(5) << endl;
```

7 10

B:

```
float a(float m, float p)
{
    return m + p;
}
```

La fonction `a` renvoie la somme des paramètres.

```
float b(float m, float p)
{
    return m * p;
}
```

La fonction `b` renvoie le produit des paramètres.

```
float c(float m)
{
    return b(m, m);
}
```

La fonction `c` renvoie le carré du paramètre.

```
...
cout << a(5, 2) << " " << b(5, 2) << endl;
→ cout << a( b(3, 2), a(5, 4) ) << endl;
cout << c(5) << endl;
```

```
7 10
```

B:

```
float a(float m, float p)
{
    return m + p;
}
```

La fonction `a` renvoie la somme des paramètres.

```
float b(float m, float p)
{
    return m * p;
}
```

La fonction `b` renvoie le produit des paramètres.

```
float c(float m)
{
    return b(m, m);
}
```

La fonction `c` renvoie le carré du paramètre.

```
...
cout << a(5, 2) << " " << b(5, 2) << endl;
cout << a( b(3, 2), a(5, 4) ) << endl;
cout << c(5) << endl;
```

6

7 10

B:

```
float a(float m, float p)
{
    return m + p;
}
```

La fonction `a` renvoie la somme des paramètres.

```
float b(float m, float p)
{
    return m * p;
}
```

La fonction `b` renvoie le produit des paramètres.

```
float c(float m)
{
    return b(m, m);
}
```

La fonction `c` renvoie le carré du paramètre.

```
...
cout << a(5, 2) << " " << b(5, 2) << endl;
→ cout << a( b(3, 2), a(5, 4) ) << endl;
cout << c(5) << endl;
```

6 9

```
7 10
```

B:

```
float a(float m, float p)
{
    return m + p;
}
```

La fonction `a` renvoie la somme des paramètres.

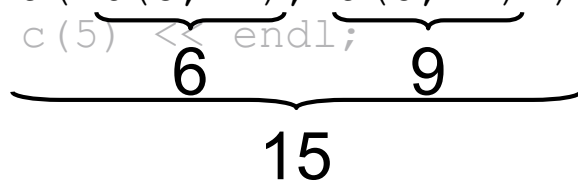
```
float b(float m, float p)
{
    return m * p;
}
```

La fonction `b` renvoie le produit des paramètres.

```
float c(float m)
{
    return b(m, m);
}
```

La fonction `c` renvoie le carré du paramètre.

```
...
cout << a(5, 2) << " " << b(5, 2) << endl;
cout << a( b(3, 2), a(5, 4) ) << endl;
cout << c(5) << endl;
```



```
7 10
```

B:

```
float a(float m, float p)
{
    return m + p;
}
```

La fonction `a` renvoie la somme des paramètres.

```
float b(float m, float p)
{
    return m * p;
}
```

La fonction `b` renvoie le produit des paramètres.

```
float c(float m)
{
    return b(m, m);
}
```

La fonction `c` renvoie le carré du paramètre.

```
...
cout << a(5, 2) << " " << b(5, 2) << endl;
cout << a( b(3, 2), a(5, 4) ) << endl;
cout << c(5) << endl;
```

```
7 10
15
```

B:

```
float a(float m, float p)
{
    return m + p;
}
```

La fonction `a` renvoie la somme des paramètres.

```
float b(float m, float p)
{
    return m * p;
}
```

La fonction `b` renvoie le produit des paramètres.

```
float c(float m)
{
    return b(m, m);
}
```

La fonction `c` renvoie le carré du paramètre.

```
...
cout << a(5, 2) << " " << b(5, 2) << endl;
cout << a( b(3, 2), a(5, 4) ) << endl;
→ cout << c(5) << endl;
```

```
7 10
15
```

B:

```
float a(float m, float p)
{
    return m + p;
}
```

La fonction `a` renvoie la somme des paramètres.

```
float b(float m, float p)
{
    return m * p;
}
```

La fonction `b` renvoie le produit des paramètres.

```
float c(float m)
{
    return b(m, m);
}
```

La fonction `c` renvoie le carré du paramètre.

```
...
cout << a(5, 2) << " " << b(5, 2) << endl;
cout << a( b(3, 2), a(5, 4) ) << endl;
→ cout << c(5) << endl;
```

```
7 10
15
25
```

Exercices

Écrivez l'en-tête puis le corps des fonctions suivantes. Pour trouver l'en-tête, il faut déterminer:

- les paramètres de la fonction, et leur type;
- le type de la fonction.

- Fonction de nom `f` qui a 1 paramètre `x`, et qui renvoie la valeur de $\sin(x) + \exp(x)$.
- Fonction de nom `degre2` qui a 4 paramètres `a`, `b`, `c` et `x`, de type `float`, et qui retourne la valeur de l'expression ax^2+bx+c
- Fonction `fact` qui renvoie la factorielle $n!$ d'un nombre entier n passé en paramètre, c'est-à-dire le produit des entiers de 1 à n .
- Fonction nommée `coeff_binomial` qui calcule le coefficient binomial de 2 nombres n et m passés en paramètre, à l'aide de la formule :

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

$n!$, $m!$ et $(n-m)!$ sont respectivement les factorielles de n , m et $(n-m)$.

$f(x)$

Fonction de nom f qui a 1 paramètre x , et qui renvoie la valeur de $\sin(x) + \exp(x)$:

```
float f(float x)
{
    float y;

    y = sin(x) + exp(x);

    return y;
}
```

ou:

```
float f(float x)
{
    return sin(x) + exp(x);
}
```

Comment utiliser (appeler) la fonction f ?

```
y = f(10); // y vaut maintenant sin(10) + exp(10)
```

```
cout << "f(0) = " << f(0) << endl;
```

degre2

Fonction de nom `degre2` qui a 4 paramètres `a`, `b`, `c` et `x`, de type `float`, et qui retourne la valeur de l'expression ax^2+bx+c

```
float degre2(float a, float b, float c, float x)
{
    float resultat;

    resultat = a * x * x + b * x + c;

    return resultat;
}
```

ou:

```
float degre2(float a, float b, float c, float x)
{
    return a * x * x + b * x + c;
}
```

Comment utiliser `degre2` ?

pour calculer par exemple $x^2 - x + 2$ en $x = 5$:

dans ce cas, les coefficients de l'expression sont: $a = 1$, $b = -1$ et $c = 2$.

```
p = degre2(1, -1, 2, 5);
```

Fonction `fact`

Fonction `fact` qui renvoie la factorielle d'un nombre entier `n` passé en paramètre, c'est-à-dire le produit des entiers de 1 à `n`.

Le produit de nombres entiers est un nombre entier.
`fact` retourne donc une valeur entière.

```
int fact(int n)
```

`n` est un *paramètre*.

On veut calculer le produit $1 \times 2 \times \dots \times n$.

```
{  
  int resultat = 1;
```

`resultat` est une *variable locale* utilisée pour calculer le produit.

```
  for(int i = 1; i <= n; i++)
```

```
    resultat = resultat * i;
```

```
  return resultat;
```

Au sortir de la boucle, `resultat` contient le produit $1 \times 2 \times \dots \times n$.

`return resultat;` permet de retourner la valeur contenue dans `resultat`.

```
}
```

Fonction `coeff_binomial`

Écrivez une fonction nommée `coeff_binomial` qui calcule le coefficient binomial de 2 nombres n et m passés en paramètre, à l'aide de la formule:

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

Pour calculer la factorielle de m , cette fonction devra appeler la fonction `fact` de la question précédente.

```
int coeff_binomial(int n, int m)
{
    return fact(n) / (fact(n - m) * fact(m));
}
```

C:

```
float g(bool c, float m, float p)
{
    if (c)
        return m;
    else
        return p;
}

...
cout << g(2 < 3, 10, 100) << endl;
if (g(3 % 2 == 0, 1, -1) < 0)
    cout << "a" << endl;
else
    cout << "b" << endl;
```

D:

```
bool h(int deb, int fin, int div)
{
    for(int i = deb; i <= fin; i++)
        if (i % div == 0)
            return true;

    return false;
}

...
if ( h(7, 13, 5) )
    cout << "oui" << endl;
else
    cout << "non" << endl;
```

C:

```
float g(bool c, float m, float p)
{
    if (c)
        return m;
    else
        return p;
}
```

...

```
→ cout << g(2 < 3, 10, 100) << endl;
```

```
if (g(3 % 2 == 0, 1, -1) < 0)
    cout << "a" << endl;
else
    cout << "b" << endl;
```

C:

```
float g(bool c, float m, float p)
{
    if (c)
        return m;
    else
        return p;
}
```

Si le premier paramètre est vrai, la fonction `g` renvoie le deuxième paramètre, sinon elle renvoie le troisième paramètre.

→ ...
`cout << g(2 < 3, 10, 100) << endl;`

```
if (g(3 % 2 == 0, 1, -1) < 0)
    cout << "a" << endl;
else
    cout << "b" << endl;
```

C:

```
float g(bool c, float m, float p)
{
    if (c)
        return m;
    else
        return p;
}
```

Si le premier paramètre est vrai, la fonction `g` renvoie le deuxième paramètre, sinon elle renvoie le troisième paramètre.

→ `cout << g(2 < 3, 10, 100) << endl;`

Le premier paramètre est vrai, donc la fonction `g` renvoie 10.

```
if (g(3 % 2 == 0, 1, -1) < 0)
    cout << "a" << endl;
else
    cout << "b" << endl;
```

10

C:

```
float g(bool c, float m, float p)
{
    if (c)
        return m;
    else
        return p;
}
```

...

```
cout << g(2 < 3, 10, 100) << endl;
```

```
→ if (g(3 % 2 == 0, 1, -1) < 0)
    cout << "a" << endl;
else
    cout << "b" << endl;
```

Si le premier paramètre est vrai, la fonction `g` renvoie le deuxième paramètre, sinon elle renvoie le troisième paramètre.

10

C:

```
float g(bool c, float m, float p)
{
    if (c)
        return m;
    else
        return p;
}
```

...

```
cout << g(2 < 3, 10, 100) << endl;
```

```
→ if (g(3 % 2 == 0, 1, -1) < 0)
    cout << "a" << endl;
else
    cout << "b" << endl;
```

Si le premier paramètre est vrai, la fonction `g` renvoie le deuxième paramètre, sinon elle renvoie le troisième paramètre.

Le premier paramètre est faux, donc la fonction `g` renvoie -1.

```
10
```

C:

```
float g(bool c, float m, float p)
{
    if (c)
        return m;
    else
        return p;
}
```

...

```
cout << g(2 < 3, 10, 100) << endl;
```

```
→ if (g(3 % 2 == 1, 1, -1) < 0)
    cout << "a" << endl;
else
    cout << "b" << endl;
```

Si le premier paramètre est vrai, la fonction `g` renvoie le deuxième paramètre, sinon elle renvoie le troisième paramètre.

Le premier paramètre est faux, donc la fonction `g` renvoie -1.

```
10
```

C:

```
float g(bool c, float m, float p)
{
    if (c)
        return m;
    else
        return p;
}
```

...

```
cout << g(2 < 3, 10, 100) << endl;
```

```
→ if (g(3 % 2 == 1, 1, -1) < 0)
    cout << "a" << endl;
else
    cout << "b" << endl;
```

Si le premier paramètre est vrai, la fonction `g` renvoie le deuxième paramètre, sinon elle renvoie le troisième paramètre.

Le premier paramètre est faux, donc la fonction `g` renvoie -1. -1 est inférieur à 0, donc la condition est vraie.

10

C:

```
float g(bool c, float m, float p)
{
    if (c)
        return m;
    else
        return p;
}
```

...

```
cout << g(2 < 3, 10, 100) << endl;
```

```
if (g(3 % 2 == 0, 1, -1) < 0)
    cout << "a" << endl;
else
    cout << "b" << endl;
```



Si le premier paramètre est vrai, la fonction `g` renvoie le deuxième paramètre, sinon elle renvoie le troisième paramètre.

Le premier paramètre est faux, donc la fonction `g` renvoie -1. -1 est inférieur à 0, donc la condition est vraie.

```
10
a
```

D:

```
bool h(int deb, int fin, int div)
{
    for(int i = deb; i <= fin; i++)
        if (i % div == 0)
            return true;

    return false;
}

if ( h(7, 13, 5) )
    cout << "oui" << endl;
else
    cout << "non" << endl;
```

D:

```
bool h(int deb, int fin, int div)
{
    for(int i = deb; i <= fin; i++)
        if (i % div == 0)
            return true;

    return false;
}
```

```
→ if ( h(7, 13, 5) )
    cout << "oui" << endl;
else
    cout << "non" << endl;
```

D:

```
bool h(int deb, int fin, int div)
{
    for(int i = deb; i <= fin; i++)
        if (i % div == 0)
            return true;

    return false;
}
```

```
→ if ( h(7, 13, 5) )
    cout << "oui" << endl;
else
    cout << "non" << endl;
```

Si un entier entre `deb` et `fin` est divisible par `div`, la fonction renvoie vrai, sinon elle renvoie faux.

D:

```
bool h(int deb, int fin, int div)
{
    for(int i = deb; i <= fin; i++)
        if (i % div == 0)
            return true;

    return false;
}
```

```
→ if ( h(7, 13, 5) )
    cout << "oui" << endl;
else
    cout << "non" << endl;
```

Si un entier entre `deb` et `fin` est divisible par `div`, la fonction renvoie vrai, sinon elle renvoie faux.

10 est entre 7 et 13, et est divisible par 5, donc `h` renvoie vrai.

D:

```
bool h(int deb, int fin, int div)
{
    for(int i = deb; i <= fin; i++)
        if (i % div == 0)
            return true;

    return false;
}
```

```
if ( h(7, 13, 5) )
    cout << "oui" << endl;
else
    cout << "non" << endl;
```

Si un entier entre `deb` et `fin` est divisible par `div`, la fonction renvoie vrai, sinon elle renvoie faux.

10 est entre 7 et 13, et est divisible par 5, donc `h` renvoie vrai.

```
oui
```