Exercice n° 1 : Retour sur les surcharges avec le pointeur et la référence

Fichier source `main.cpp`

```
1   #include <iostream>
2   using namespace std;
3
4   #include "Point.h"
5
6   int main ( )
7   {
8       Point p1 ( 1, 2 );
9       Point p2;
10      Point *p3;
11
12      p2 = p1.M1 ( );
13      cout << "p1 = " << p1 << endl;
14      cout << "p2 = " << p2 << endl;
15
16      p3 = p2.M2 ( );
17      cout << "p2 = " << p2 << endl;
18      cout << "*p3= " << *p3 << endl;
19
20      p1 = p2.M3 ( );
21      cout << "p2 = " << p2 << endl;
22      cout << "p1 = " << p1 << endl;
23
24      return 0;
25  }
```

Fichier source `Point.h`

```
1   #if ! defined (POINT_H)
2   #define POINT_H
3
4   #include <iostream>
5   using namespace std;
6
7   class Point
8   {
9     public :
10      friend ostream & operator << ( ostream & out, const Point & p );
11
12      Point   M1 ( );
13      Point * M2 ( );
14      Point & M3 ( );
15
16      Point & operator ++ ( void );
17      Point operator ++ ( int inutile );
18
19      Point & operator = ( const Point & p );
20
21      Point ( const Point & p );
22      Point ( int abs = 0, int ord = 0 );
23      ~Point ( );
24
25    private :
26      int x;
27      int y;
28  };
29
30  #endif
```

Fichier source `Point.cpp`

```
1   #include <iostream>
2   using namespace std;
3
4   #include "Point.h"
5
6   Point Point::M1 ( )
7   {
8   #ifdef MAP
9       cout << ">> M1 : retour par valeur d'un Point" << endl;
10  #endif
11      (*this)++;
12      return *this;
13  }
14
15  Point * Point::M2 ( )
16  {
17  #ifdef MAP
18    cout << ">> M2 : retour par valeur d'un pointeur sur Point" << endl;
19  #endif
20    ++(*this);
21    return this;
22  }
23
24  Point & Point::M3 ( )
25  {
26  #ifdef MAP
27      cout << ">> M3 : retour par référence d'un Point" << endl;
28  #endif
29      (*this)++;
30      return *this;
31  }
32
33  ostream & operator << ( ostream & out, const Point & p )
34  {
35      out << "( " << p.x << "," << p.y << " )";
36      return out;
37  }
38
39  Point & Point::operator ++ ( void )
40  {
41      ++x;
42      ++y;
43      return *this;
44  }
45
46  Point Point::operator ++ ( int inutile )
47  {
48      Point tmp ( x, y );
49      x++;
50      y++;
51      return tmp;
52  }
53
```

Fichier source `Point.cpp` (suite)

```
54  Point & Point::operator = ( const Point & p )
55  {
56      if ( this != &p )
57      {
58          x = p.x;
59          y = p.y;
60      }
61      return *this;
62  }
63
64  Point::Point ( const Point & p )
65  {
66  #ifdef MAP
67      cout << ">> Constructeur de copie de la classe <Point>" << endl;
68  #endif
69      x = p.x;
70      y = p.y;
71  }
72
73  Point::Point ( int abs, int ord )
74  {
75  #ifdef MAP
76      cout << ">> Constructeur de la classe <Point>" << endl;
77  #endif
78      x = abs;
79      y = ord;
80  }
81
82  Point::~Point ( )
83  {
84  #ifdef MAP
85      cout << ">> Destructeur de la classe <Point>" << endl;
86  #endif
87  }
88
```

1. Donner les notations fonctionnelles pour les écritures suivantes :

```
p2 = p1.M1 ( ); // ligne 12 du fichier main.cpp
p3 = p2.M2 ( ); // ligne 16 du fichier main.cpp
(*this)++;      // ligne 11 du fichier Point.cpp
++(*this);      // ligne 20 du fichier Point.cpp
++p++;
cout << "p1 = " << p1 << endl; // ligne 13 du fichier main.cpp
```

2. Donner la notation fonctionnelle pour la ligne : `cout << p++ << endl;` Cela correspond à la ligne 10 de la fonction **`main`** ci-dessous.

Fichier source `main.cpp`

```
1   #include <iostream>
2   using namespace std;
3
4   #include "Point.h"
5
6   int main ( )
7   {
8       Point p ( 1, 2 );
9
10      cout << p++ << endl;
11      cout << p << endl;
12
13      return 0;
14  }
```

3. Faire le dessin mémoire pour la simulation de l'exécution de cette fonction simplifiée du **`main`**. On arrêtera la simulation à la ligne 11, en excluant l'exécution de cette ligne.

4. Donner la trace exacte à l'écran, si elle existe, pour la fonction ordinaire **`main`** ci-dessus. On suppose que la constante **`MAP`** est définie à la compilation (ajout des lignes en compilation conditionnelle).