Blocs contenant une seule instruction

Jusqu'à présent, à chaque fois que nous écrivions une structure (boucle ou condition), nous utilisions les caractères `{` et `}` pour délimiter un bloc contenant les instructions de la structure. Cela donnait donc des programmes de ce type :

```
if (hauteurPlongeoir <= 3)
{
   cout << "On saute !" << endl;
}
else
{
   cout << "J'ai le vertige…!" << endl;
}
```

Comme nous l'avions vu rapidement, si l'on enlève les accolades, alors on ne peut rattacher qu'une instruction à la boucle ou la condition ; mais on la rattache ! On peut donc écrire le programme précédent de cette façon :

```
if (hauteurPlongeoir <= 3)
   cout << "On saute !" << endl;
else
   cout << "J'ai le vertige…!" << endl;
```

Nous nous permettrons désormais de présenter des codes avec des instructions seules à la place des blocs afin de réduire l'écriture. Si vous le faites aussi, prenez garde aux étourderies !

Blocs contenant uniquement une structure

La règle ci-dessus marche également si un bloc ne contient pas une seule ligne de code, mais une seule structure. Ainsi, le code :

```
for (int iLigne = 1; iLigne <= 10; iLigne = iLigne + 1)
{
   for (int iCol = 1; iCol <= 10; iCol = iCol + 1)
   {
      if (iLigne + iCol == 11)
      {
         cout << iLigne << ' ' << iCol << endl;
      }
   }
}
```

est équivalent au code suivant :

```
for (int iLigne = 1; iLigne <= 10; iLigne = iLigne + 1)
   for (int iCol = 1; iCol <= 10; iCol = iCol + 1)
      if (iLigne + iCol == 11)
         cout << iLigne << ' ' << iCol << endl;
```

On pourrait aussi écrire l'algorithme suivant :

```
Pour iLigne dans [1 ; 10]
   Pour iCol dans [1 ; 10]
      Si iLigne + iCol = 11
         Afficher "X" (sans retour à la ligne)
      Sinon
         Afficher " " (sans retour à la ligne)
   Aller à la ligne
```

de cette façon :

```
for (int iLigne = 1; iLigne <= 10; iLigne = iLigne + 1)
{
   for (int iCol = 1; iCol <= 10; iCol = iCol + 1)
      if (iLigne + iCol == 11)
         cout << "X";
      else
         cout << " ";
   cout << endl;
}
```

Le `if...else` est donc considéré comme une seul instruction, le `else` étant associé au dernier `if` rencontré. Remarque : si l'on a un `if` qui contient un `if`, puis qui est suivi d'un `else`, que peut-il se passer suivant la position ou l'absence des accolades ?

Il était obligatoire ici de laisser les délimiteurs restants pour que l'affichage du retour à la ligne soit au sein de la boucle extérieure.

Explications sur le `else if`

Considérons le code suivant :

```
if (prix >= 300)
{
   prix = prix - 40;
}
else
{
   if (prix >= 200)
   {
      prix = prix - 25;
   }
   else
   {
      if (prix >= 100)
      {
         prix = prix - 10;
      }
   }
}
```

alors, vous savez désormais qu'il est équivalent à :

```
if (prix >= 300)
   prix = prix - 40;
else
   if (prix >= 200)
      prix = prix - 25;
   else
      if (prix >= 100)
         prix = prix - 10;
```

et donc (puisque l'identation est juste visuelle ici, sans signification) à :

```
if (prix >= 300)
   prix = prix - 40;
else if (prix >= 200)
   prix = prix - 25;
else if (prix >= 100)
```

```
    prix = prix - 10;
```

ce qui est donc bien équivalent à la manière dont nous avons utilisé le `else if` précédemment. Ainsi, la forme `else if` est simplement l'imbrication d'un `if` dans un `else` dont on a enlevé les accolades superflues afin de simplifier l'écriture.