Dans un précédent cours nous avons vu comment calculer un arrondi **à l'entier inférieur** ou **à l'entier supérieur**. Nous allons maintenant voir comment calculer un arrondi **au plus proche** pour lequel, de manière naturelle, la valeur 1.3 sera arrondie à l'entier 1 et la valeur 1.8 sera arrondie à l'entier 2.

Il ne faut pas oublier d'importer la bibliothèque de maths (comme pour les fonctions `floor()` et `ceil()`), ensuite il suffit d'utiliser la fonction `round()` :

```
cout << round(1.3) << endl;
cout << round(1.8) << endl;
```

↳
```
1
2
```

Pour se souvenir du nom de cette fonction, penser que "round" est la traduction en anglais du mot "arrondi".

Si on ne programme pas sous linux (système d'exploitation utilisé pour évaluer les programmes sur ce site), il est possible que la fonction `round()` ne soit pas disponible. En effet, elle ne fait pas officiellement partie du standard C++ avant sa version C++11. Si la fonction `round()` n'est pas disponible, vous pouvez utiliser le code suivant :

```
cout << floor(1.3 + 0.5) << endl;
cout << floor(1.8 + 0.5) << endl;
```

↳
```
1
2
```

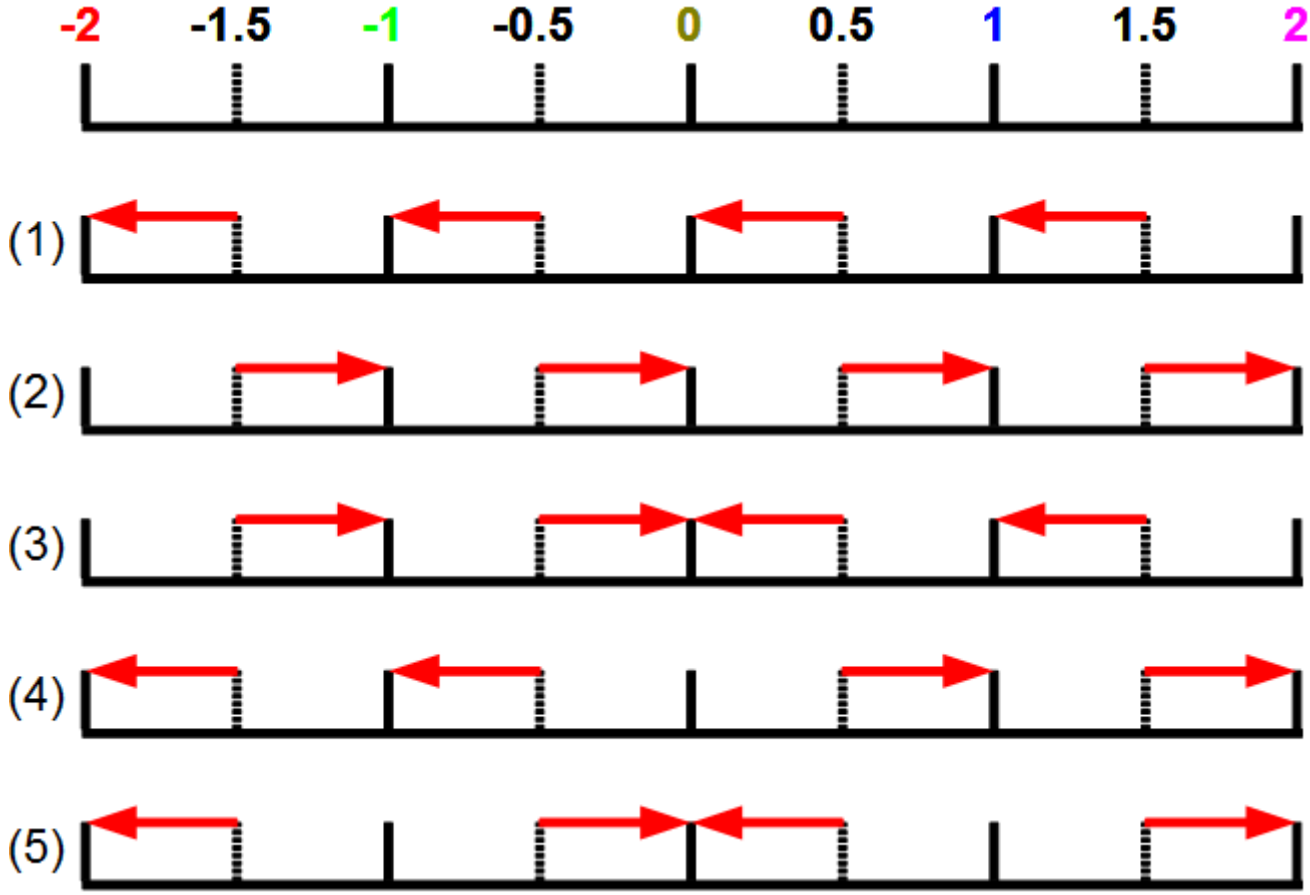
Se convaincre que cela fait bien un arrondi au plus proche n'est pas très difficile, essayez de le montrer.

Cas du 0.5 : différentes possibilités

Comment arrondir la valeur 1.5 ? Faut-il l'arrondir à 1 ou à 2 ? Il existe plusieurs choix possibles avec tous une bonne justification. Voici les principaux :

1. Au plus petit : si deux choix sont possibles, prendre le plus petit.
2. Au plus grand : si deux choix sont possibles, prendre le plus grand.
3. Vers zéro : si deux choix sont possibles, prendre le plus proche de zéro.
4. Vers l'infini : si deux choix sont possibles, prendre le plus éloigné de zéro.
5. Bancaire : si deux choix sont possibles, prendre celui qui est pair.

Et en voici une illustration visuelle



L'arrondi bancaire est appelé ainsi car utilisé par les banques. En effet si on ne regarde que les nombres positifs, tous les autres arrondis arrondissent toujours dans la même direction : des petites erreurs pourraient s'accumuler. En choisissant un arrondi qui va changer de direction selon la parité les erreurs ne s'accumulent plus et vont se compenser naturellement.

Notez qu'en mathématique ces arrondis sont également utilisés, parfois sous un nom différent :

- Si rien n'est précisé, il s'agit d'un arrondi "au plus grand".
- Si on parle d'un arrondi "par défaut", il s'agit d'un arrondi "au plus petit".
- Si on parle d'un arrondi "par excès", il s'agit d'un arrondi "au plus grand".
- Si on parle de valeur tronquée (on enlève tout ce qui est après la virgule), il s'agit d'un arrondi "vers zéro".

Cas du 0.5 : ce que fait C++

Alors, quelle est la technique utilisée en C++ ? Regardons sur quelques valeurs !

```
cout << round(-1.5) << endl;
cout << round(-0.5) << endl;
cout << round(0.5) << endl;
cout << round(1.5) << endl;
```

↳

```
-2
-1
1
2
```

Il s'agit donc d'un arrondi "vers l'infini" quand il y a deux choix possibles.

Mais si vous n'avez pas la fonction `round()` disponible et que vous utilisez `floor()` :

```
cout << floor(-1.5 + 0.5) << endl;
```

```
cout << floor(-0.5 + 0.5) << endl;
cout << floor(0.5 + 0.5) << endl;
cout << floor(1.5 + 0.5) << endl;
```

↳

```
-1
0
1
2
```

Il s'agit donc d'un arrondi "au plus grand" quand il y a deux choix possibles.

En pratique le type d'arrondi utilisé pour calculer un arrondi au plus proche importe peu puisque les deux valeurs sont possibles, il s'agit juste d'une convention qui est prise dans un langage de programmation donné (ou en mathématique). Les exercices sur le site sont insensibles au type d'arrondi au plus proche utilisé.