Imaginons qu'on souhaite lire plusieurs entiers et afficher uniquement la dernière valeur. On pourrait penser au programme **faux** suivant :

```
int nbValeurs;
cin >> nbValeurs;
repeat (nbValeurs)
{
   int laValeur;
   cin >> laValeur;
}
cout << laValeur << endl;
```

Quand on essaie d'exécuter ce programme, on obtient une erreur :

```
In function 'int main()':
line 14: error: 'laValeur' was not declared in this scope
```

Qu'est-ce que cela signifie ? Le langage C++ limite la *portée* des variables au bloc dans lequel elles ont été déclarées, c'est-à-dire que quand on définit une variable, celle-ci devient inaccessible quand l'exécution du bloc est terminée : **elle n'existe plus quand on quitte le bloc**.

Ainsi, ci-dessus, la variable *laValeur* est définie dans le bloc d'une boucle ; quand le bloc se termine, c'est-à-dire **à la fin de chaque tour de boucle**, la variable disparaît. Si on effectue un nouveau tour de boucle, c'est alors une nouvelle variable *laValeur*, indépendante de la précédente, qui est créée. Après la boucle (l'accolade fermante `}`), il n'existe donc plus de variable *laValeur*.

Remarquez par ailleurs que si 0 est saisi pour *nbValeurs*, aucun tour de boucle ne va être effectué ! Il devient donc logique de ne pas pouvoir utiliser la variable *laValeur*.

Pour que le programme précédent fonctionne, il faut déclarer cette variable avant la boucle. On peut utiliser une valeur particulière pour représenter le fait qu'aucune variable n'a été lue. Dans le programme corrigé ci-dessous, on a choisi la valeur -1 :

```
int nbValeurs;
cin >> nbValeurs;
int laValeur = -1;
repeat (nbValeurs)
{
   cin >> laValeur;
}
cout << laValeur << endl;
```

Portée et variables de même nom

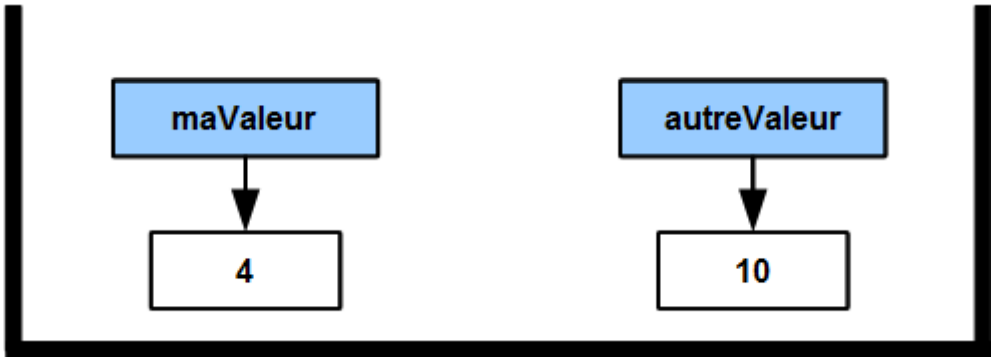
Même si cela n'est pas recommandé, en C++, il est possible qu'une variable définie dans un bloc ait le même nom qu'une variable définie en dehors de ce bloc. Par exemple, dans le code suivant, la variable *maValeur* est dans ce cas :

```
int maValeur = 4;
int autreValeur = 10;
repeat (1)
{
   int maValeur = 2;
   int uneValeur = 5;
   cout << autreValeur << " " << maValeur << endl;
}
cout << autreValeur << " " << maValeur << endl;
```

```
10 2
10 4
```

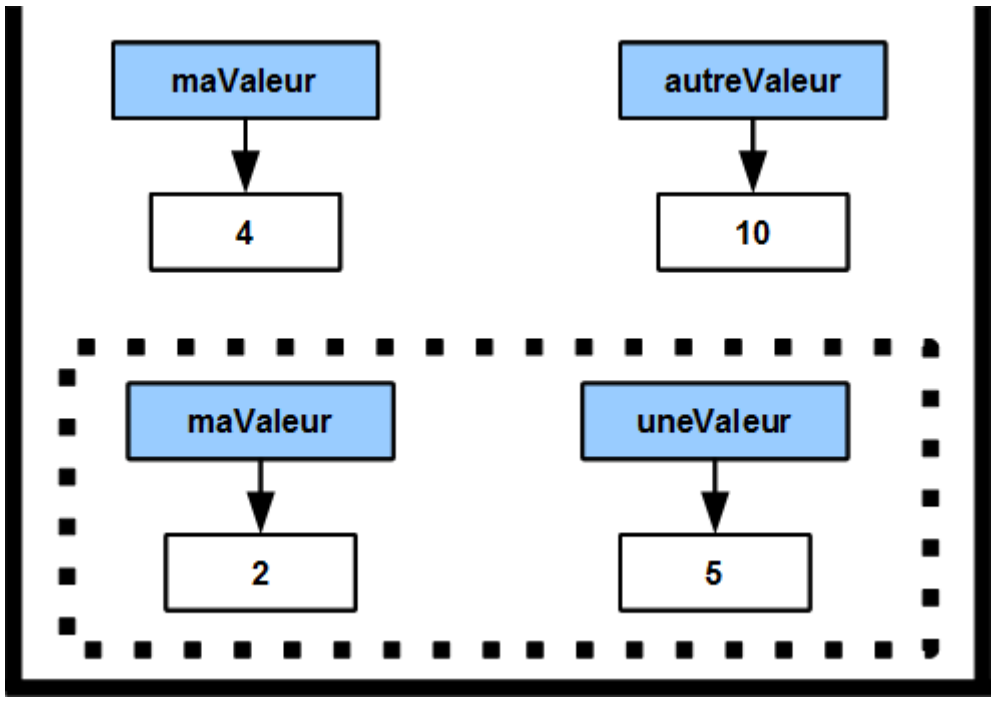
Intéressant, non ? Quand on a créé la variable *maValeur* valant 2 à l'intérieur de la boucle, la précédente qui valait 4 n'a pas été écrasée, elle a juste été mise de côté tant qu'on était dans le bloc pour laisser la place à la nouvelle (et donc on a affiché la valeur 2) puis elle est « revenue » à la fin du bloc.

Représentons les variables pour nous aider à y voir plus clair. Avant de rentrer dans la boucle, on a deux variables, et *maValeur* vaut 4 :

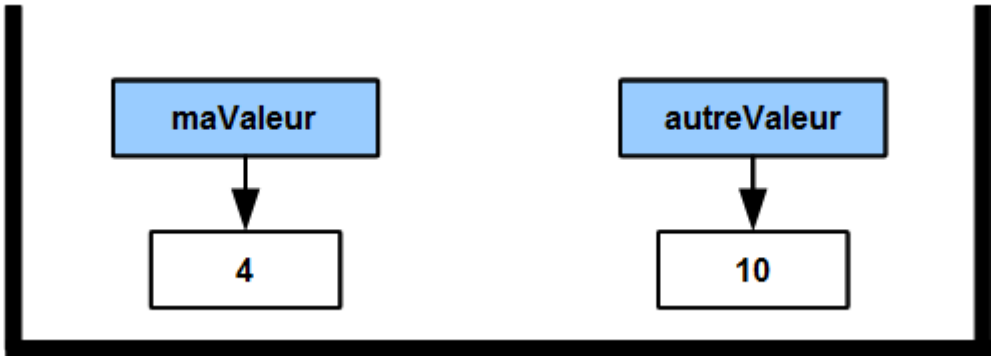


Sur l'image suivante, on a placé l'ensemble des variables définies dans le bloc (c'est-à-dire la boucle) dans un cadre en pointillés. Une fois dans le bloc, la variable *maValeur* valant 4 n'a pas disparu, mais on a une nouvelle variable *maValeur* valant 2 qui n'existe que dans le bloc.

Quand on cherche la valeur d'une variable, on va d'abord regarder si elle existe dans le bloc, auquel cas on s'arrête. Sinon, on va regarder en dehors du bloc. Ainsi, pour *maValeur*, on aura la valeur 2 (car une variable *maValeur* est définie dans le bloc avec la valeur 2), et pour *autreValeur*, on aura la valeur 10 (car aucune variable *autreValeur* n'est définie dans le bloc) :



Après la boucle, on revient à la situation du début avec une seule variable *maValeur* qui vaut 4 :



Par conséquent, cette histoire de « portée » n'étant pas si simple, on préférera tout simplement utiliser toujours le même nom pour une variable définie dans un bloc et une variable définie dans un bloc supérieur.

En revanche, il est tout à fait possible (et même fréquent) de déclarer deux variables de même nom dans deux blocs différents. Par exemple dans ce programme, qui affiche le double de deux nombres saisis, puis le carré de trois autres nombres :

```
repeat (2)
{
   int nombre;
   cin >> nombre;
   cout << (nombre * 2) << endl;
}
repeat (3)
{
   int nombre;
   cin >> nombre;
   cout << (nombre * nombre) << endl;
}
```

On déclare ainsi une variable *nombre* à deux endroits différents, dans des blocs complètement séparés, et on en fait deux utilisations indépendantes.

En fait, il y a même plus de portées indépendantes de la variable *nombre* : *nombre* est déclarée à chaque tour de boucle quand l'instruction est traversée, et cela crée une nouvelle variable qui ne partage aucune valeur avec les *nombre* des autres tours de boucle.

Si l'on veut réutiliser une variable en conservant sa valeur au fur et à mesure des répétitions, il faut la déclarer avant la boucle, comme on l'a déjà fait dans certains problèmes (par exemple Invasion de batraciens).