- Solution

The solution to the exercise of the previous lesson will be discussed in this lesson.

WE'LL COVER THE FOLLOWING

Solution

```cpp
// perfectForwarding.cpp
#include <string>
#include <utility>
#include <initializer_list>
#include <vector>
#include <iostream>

template <typename T, typename ... Arg>
T createT(Arg&& ... arg){
  return T(std::forward<Arg>(arg) ... );
}

int main(){

  int lValue= createT<int>(1);
  int i= createT<int>(lValue);
  std::cout << "lvalue = " << lValue;
  std::cout<< " " <<std::endl;

  std::cout << "i = " << i;
  std::cout<< " " <<std::endl;

  std::string s= createT<std::string>("Only for testing purpose.");
  std::cout << s;
  std::cout<< " " <<std::endl;

  typedef std::vector<int> IntVec;
  IntVec intVec= createT<IntVec>(std::initializer_list<int>({1, 2, 3, 4, 5}));
    for (auto i = intVec.begin(); i != intVec.end(); ++i)
    std::cout << *i << " ";
}
```

Explanation

- The three dots in line 10 `std::forward<Args>(args)...` cause each constructor call to perform perfect forwarding. The result is impressive.
- Now, we can invoke the perfect factory method with the number of arguments of our choice, as seen in lines 16, 23 and 28.

For further information, read perfect forwarding by Thomas Becker.

In the next lesson, we will study the first section of memory management: memory allocation.