```cpp
#include <iostream>
int main() {
    using namespace std;
    int arr[] = { 1,2,3,4,5 };
    // Normal for loop
    for (int i = 0; i < 5; i++) {
        cout << arr[i] << " ";
    }
    // Range-based for loop syntax
    for (variable declaration: range){
        statements
    }
    // For example:
    for (int x : arr)
    {
        cout << x << " ";
    }
    // This example is somewhat naive. This is because for each iteration a COPY of the variable will be created. to avoid this tyou can use a reference.
    for (int& x : arr)
    {
        cout << x << " ";
    }
    // To make it even more precise, you can write it as
    // const int. But since you know that the compiler would
    // know that it is of type int, so we can change the word
    // int to the word auto.
    for (const auto & x : arr) {

        cout << x << " ";
    }
    for (auto x : { 1,2,3,4 }) {
```

```
32          cout << x;
33      }
34      return 0;
35
36      // How the range-based for loop works under
          the hood:
37      // Create a pointer to the beginning of the
          container
38      int* beg = std::begin(arr), *end =
          std::end(arr);
39      // std::begin and std::end return iterators,
          which are pointers, so this operation is
          typesafe. And because we know the return
          value of std::begin and std::end, then we
          can automatically infer the types of beg
          and end. So:
40      auto beg = std::begin(arr);
41      auto end = std::end(arr);
42      // In C++ 17, they assert that begin and end
          must be of the same type.
43      for (; beg!=end; ++beg){
44          auto x = *beg;
45      }
46      // Forwarding Reference: (KIV)
47      // Forwarding references are a special kind
          of references that preserve the value
          category of a function argument, making it
          possible to FORWARD it by means of
          std::forward. Forwarding references are
          either:
48
49      // 1) Function parameter of a function
          template declared as rvalue reference to cv-
          unqualified typetemplate parameter of that
          same function type.
50      // auto&& except when deduced from a brace-
          enclosed initialiser list.
51      auto&& range = arr; //type of range should be
```

```
51          the same as the type of array.
52  }
53
```