```cpp
#include <iostream>
#include "ex03.h"


// NOTE:
// I have changed names to shorter ones to make code more compact.


// Do not modify
CurrencyConverter::CurrencyConverter() {
    currencies.insert("DKK");
    currencyToExchangeRate["DKK"] = 1;
}


// Do not modify
bool CurrencyConverter::supportsCurrency(std::string currency) {
    if (currencies.find(currency) != currencies.end())
        return true;
    else
        return false;
}



////////////////////////////////////////////////////////////////////////////
// Exercise 3 (a) Check and correct if necessary
////////////////////////////////////////////////////////////////////////////

// ORIGINAL CODE:
/*void CurrencyConverter::print() {
    cout << "The converter supports the following currencies:" << endl;
    for (map<string, double>::iterator it = currencyToExchangeRate.begin();
         it != currencyToExchangeRate.end(); ++it) {
        cout << ' ' << "currency " << it->second << " has exchange rate "
             << it->first << endl;
    }
}*/

//**************************************************************************
// IMPROVEMENTS:
// The verbose declaration of the iterator can be omitted in the for loop.
// To be more concise and elegant, a range-based for loop is even better.

// ERRORS:
// - The single character delimiter in cout.
// - The map's key and value are printed in reversed order.
//**************************************************************************

// IMPROVED CODE:

void CurrencyConverter::print() {
    std::cout << "Supported currencies:" << std::endl;
    for (auto & c : currencyToExchangeRate) {
        std::cout << ' ' << "currency " << c.first << " has exchange rate ";
        std::cout << c.second << std::endl;
    }
}
```

```
59
60  ////////////////////////////////////////////////////////////////////////////
61  // Exercise 3 (b) Check and correct if necessary
62  ////////////////////////////////////////////////////////////////////////////
63
64
65  // ORIGINAL CODE
66  /*bool CurrencyConverter::addCurrency(string currency, double rateToDKK) {
67      if (supportsCurrency(currency)) {
68          //I already have this element. Hence I return false
69          return false;
70      }
71      else if (rateToDKK <= 0) {
72          //Exchange rates must be positive
73          return false;
74      }
75      else {
76          currencies.insert(currency);
77          currencyToExchangeRate[rateToDKK] = currency;  // ERROR
78          return true;
79      }
80  }*/
81
82
83  //***************************************************************************
84  // ERROR: 'exchangeRateToDKK' and 'currencyCode' are swapped.
85  //***************************************************************************
86
87
88  // IMPROVED CODE:
89
90  bool CurrencyConverter::addCurrency(std::string currency, double rateToDKK) {
91      if (supportsCurrency(currency))
92          return false;
93      else if (rateToDKK <= 0)
94          return false;
95      else {
96          currencies.insert(currency);
97          currencyToExchangeRate[currency] = rateToDKK;
98          return true;
99      }
100 }
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
```

```cpp
////////////////////////////////////////////////////////////////////////
// Exercise 3 (c) Implement this function
////////////////////////////////////////////////////////////////////////

// Update the currency exchange rate for an existing currency.
// Though at the exam this would've caused a fail because it was evaluated by
// an automatic test which didn't expect it, it's nice to prompt an error
// message.

bool CurrencyConverter::updateExchangeRate(std::string currency, double newRate) {
    if (!supportsCurrency(currency)) {
        std::cout << "Currency not supported" << std::endl;
        return false;
    }
    else if (newRate <= 0) {
        std::cout << "Invalid (negative) exchange rate" << std::endl;
        return false;
    }
    else if (currency == "DKK") {
        std::cout << "Cannot update exhange rate for DKK" << std::endl;
        return false;
    }
    else {
        currencyToExchangeRate[currency] = newRate;
        return true;
    }
}



// NOTE:
// The next 2 methods do exactly the same thing except for the operation to
// perform. So, I added an internal method to avoid repeating code.
// To specify the operation to perform I use a lambda function.


////////////////////////////////////////////////////////////////////////
// Exercise 3 (d) Implement this function
////////////////////////////////////////////////////////////////////////

// Convert to DKK.

double CurrencyConverter::convertToDKK(double amount, std::string currency) {
    return serve(amount , currency , [](double x , double y){ return x * y; });
}



////////////////////////////////////////////////////////////////////////
// Exercise 3 (e) Implement this function
////////////////////////////////////////////////////////////////////////

// Convert from DKK to another given currency.

double CurrencyConverter::convertFromDKK(double amountDKK, std::string currency) {
    return serve(amountDKK , currency , [](double x , double y){ return x / y; });
}

```

```
175
176 double CurrencyConverter::serve(double x, std::string s, double (*f)(double, double)) {
177     if (x <= 0)
178         return -1;
179     else if (!supportsCurrency(s))
180         return -1;
181     else
182         return f(x, currencyToExchangeRate[s]);
183 }
```