Design a function template named compare that takes two arguments of the same type and returns a boolean value indicating whether the first argument is greater than, less than, or equal to the second argument. How would you adapt this template to work with custom data types?

Implement a function template named swap that exchanges the values of two variables of the same type. Discuss the potential limitations of this approach when dealing with complex data structures.

Consider a scenario where you need to find the minimum value in an array. Create a function template named findMin that works with any data type for which the comparison operator (<) is defined. Explain how function templates promote code reusability in this case.

#include<iostream>

template <typename T>

bool compare(T a, T b, char op) {

switch (op) {

case '>':

return a > b;

case '<':

return a < b;

case '=':

return a == b;

default:

throw std::invalid\_argument("Invalid operator");

}

}

template <typename T>

void swap(T& a, T& b) {

T temp = a;

a = b;

b = temp;

}

int main() {

int x = 5;

int y = 10;

std::cout << "Before swap: x = " << x << ", y = " << y << std::endl;

swap(x, y);

std::cout << "After swap: x = " << x << ", y = " << y << std::endl;

return 0;

}

Output:

