```cpp
//Q. Implement a custom dynamic array class that supports basic operations
//like insertion, deletion, resizing, and clearing.
#include <iostream>
#include <stdexcept>
#include <algorithm>

template <typename T>
class DynamicArray {
public:
   DynamicArray();
   ~DynamicArray();
   DynamicArray(const DynamicArray& other);
   DynamicArray& operator=(const DynamicArray& other);

   void insert(size_t index, const T& value);
   void remove(size_t index);
   void clear();
   size_t size() const;
   T& operator[](size_t index);
   const T& operator[](size_t index) const;

private:
   void resize(size_t newCapacity);

   T* data;
   size_t currentSize;
   size_t capacity;

   static const size_t INITIAL_CAPACITY = 10;
};

template <typename T>
DynamicArray<T>::DynamicArray()
   : data(new T[INITIAL_CAPACITY]), currentSize(0),
capacity(INITIAL_CAPACITY) {}

template <typename T>
DynamicArray<T>::~DynamicArray() {
   delete[] data;
}
```

```cpp
template <typename T>
DynamicArray<T>::DynamicArray(const DynamicArray& other)
   : data(new T[other.capacity]), currentSize(other.currentSize),
capacity(other.capacity) {
   std::copy(other.data, other.data + other.currentSize, data);
}

template <typename T>
DynamicArray<T>& DynamicArray<T>::operator=(const DynamicArray& other) {
   if (this != &other) {
       T* newData = new T[other.capacity];
       std::copy(other.data, other.data + other.currentSize, newData);

       delete[] data;

       data = newData;
       currentSize = other.currentSize;
       capacity = other.capacity;
   }
   return *this;
}

template <typename T>
void DynamicArray<T>::insert(size_t index, const T& value) {
   if (index > currentSize) {
       throw std::out_of_range("Index out of range");
   }

   if (currentSize == capacity) {
       resize(capacity * 2);
   }

   for (size_t i = currentSize; i > index; --i) {
       data[i] = data[i - 1];
   }
   data[index] = value;
   ++currentSize;
}
```

```cpp
template <typename T>
void DynamicArray<T>::remove(size_t index) {
    if (index >= currentSize) {
        throw std::out_of_range("Index out of range");
    }

    for (size_t i = index; i < currentSize - 1; ++i) {
        data[i] = data[i + 1];
    }
    --currentSize;
}

template <typename T>
void DynamicArray<T>::clear() {
    currentSize = 0;
}

template <typename T>
size_t DynamicArray<T>::size() const {
    return currentSize;
}

template <typename T>
T& DynamicArray<T>::operator[](size_t index) {
    if (index >= currentSize) {
        throw std::out_of_range("Index out of range");
    }
    return data[index];
}

template <typename T>
const T& DynamicArray<T>::operator[](size_t index) const {
    if (index >= currentSize) {
        throw std::out_of_range("Index out of range");
    }
    return data[index];
}

template <typename T>
void DynamicArray<T>::resize(size_t newCapacity) {
```

```
    T* newData = new T[newCapacity];
    std::copy(data, data + currentSize, newData);

    delete[] data;

    data = newData;
    capacity = newCapacity;
}

int main() {

    DynamicArray<int> arr;

    arr.insert(0, 1);
    arr.insert(1, 2);
    arr.insert(2, 3);
    arr.insert(1, 4);

    std::cout << "Array contents: ";
    for (size_t i = 0; i < arr.size(); ++i) {
        std::cout << arr[i] << " ";
    }
    std::cout << std::endl;

    arr.remove(2);

    std::cout << "Array contents after removal: ";
    for (size_t i = 0; i < arr.size(); ++i) {
        std::cout << arr[i] << " ";
    }
    std::cout << std::endl;

    arr.clear();

    std::cout << "Array size after clearing: " << arr.size() << std::endl;

    return 0;
}
```

Output
rps@rps-virtual-machine:~/Desktop/C Demo$ cd "/home/rps/Desktop/C Demo/" && g++ first.cpp -o first && "/home/rps/Desktop/C Demo/"first
Array contents: 1 4 2 3
Array contents after removal: 1 4 3
Array size after clearing: 0
rps@rps-virtual-machine:~/Desktop/C Demo$ ^C
rps@rps-virtual-machine:~/Desktop/C Demo$

Q. Create a template-based stack class supporting push, pop, and peek operations. Implement it for different data types like int, float, and std::string.

```
#include <iostream>
#include <vector>
#include <stdexcept>

template <typename T>
class Stack {
private:
   std::vector<T> data;
public:
   void push(const T& value) {
       data.push_back(value);
   }

   void pop() {
       if (data.empty()) {
           throw std::out_of_range("Stack is empty");
       }
       data.pop_back();
   }

   T& peek() {
       if (data.empty()) {
           throw std::out_of_range("Stack is empty");
```

```
        }
        return data.back();
    }

    bool isEmpty() const {
        return data.empty();
    }
};

int main() {
    Stack<int> intStack;
    intStack.push(1);
    intStack.push(2);
    std::cout << "Top of intStack: " << intStack.peek() << std::endl;
    intStack.pop();
    std::cout << "Top of intStack after pop: " << intStack.peek() <<
std::endl;

    Stack<float> floatStack;
    floatStack.push(1.1f);
    floatStack.push(2.2f);
    std::cout << "Top of floatStack: " << floatStack.peek() << std::endl;
    floatStack.pop();
    std::cout << "Top of floatStack after pop: " << floatStack.peek() <<
std::endl;

    Stack<std::string> stringStack;
    stringStack.push("Hello");
    stringStack.push("World");
    std::cout << "Top of stringStack: " << stringStack.peek() << std::endl;
    stringStack.pop();
    std::cout << "Top of stringStack after pop: " << stringStack.peek() <<
std::endl;

    return 0;
}
```

Output

```
rps@rps-virtual-machine:~/Desktop/C Demo$ cd "/home/rps/Desktop/C Demo/" && g++ second.cpp -o second && "/home/rps/Desktop/C Demo/"second
Top of intStack: 2
Top of intStack after pop: 1
Top of floatStack: 2.2
Top of floatStack after pop: 1.1
Top of stringStack: World
Top of stringStack after pop: Hello
rps@rps-virtual-machine:~/Desktop/C Demo$
```

Q. Write a program that reads from a file and handles various exceptions such as file not found, read errors, and unexpected data formats.

```cpp
#include <iostream>
#include <fstream>
#include <string>
#include <sstream>
#include <exception>
using namespace std;

void readFile(const string& filename) {
    ifstream file(filename);
    if (!file.is_open()) {
        throw runtime_error("File not found or could not be opened.");
    }

    string line;
    while (getline(file, line)) {
        istringstream iss(line);
        int value;
        if (!(iss >> value)) {
            throw runtime_error("Unexpected data format encountered.");
        }
        cout << "Read value: " << value << endl;
    }

    if (file.bad()) {
        throw runtime_error("Error occurred while reading the file.");
```

```
    }
}

int main() {
    const string filename = "data.txt";

    try {
        readFile(filename);
    } catch (const runtime_error& e) {
        cerr << "Runtime error: " << e.what() << endl;
        return 1;
    } catch (const exception& e) {
        cerr << "Exception: " << e.what() << endl;
        return 1;
    } catch (...) {
        cerr << "Unknown exception occurred." << endl;
        return 1;
    }

    return 0;
}
```

Output:

```
rps@rps-virtual-machine:~/Desktop/C Demo$ cd "/home/rps/Desktop/C Demo/" && g++
third.cpp -o third && "/home/rps/Desktop/C Demo/"third
Runtime error: Unexpected data format encountered.
rps@rps-virtual-machine:~/Desktop/C Demo$ cd "/home/rps/Desktop/C Demo/" && g++
third.cpp -o third && "/home/rps/Desktop/C Demo/"third
Runtime error: Unexpected data format encountered.
rps@rps-virtual-machine:~/Desktop/C Demo$ cd "/home/rps/Desktop/C Demo/" && g++
third.cpp -o third && "/home/rps/Desktop/C Demo/"third
Read value: 1
Read value: 2
Read value: 3
Read value: 4
Read value: 5
Read value: 6
Read value: -9
Read value: 7
```

Read value: 8
Read value: 400
rps@rps-virtual-machine:~/Desktop/C Demo$

Q. Write a unit test suite for the custom dynamic array class using a testing framework like Google Test or CppUnit.