6.1

6.1 Explain what is fundamentally wrong with structuring the code for a function as shown below.

```
void func() {
    initialize(); // perform initialization
    do_work(); // do some work
    cleanup(); // perform any necessary cleanup
}
```

This function does not handle if the `do_work()` functions throw any exception, the `cleanup()` function will never be called. Probably cause resource leak.

6.2

6.2 Consider the code given below, in which `Thing` is some class type. Can the function `analyze` throw an exception? Justify your answer. Can the function `doWork` throw an exception? Justify your answer.

```
// The type Thing is defined in the following header file.
#include <Thing.hpp>

Thing globalThing;

void analyze(Thing x) noexcept
{
    // ...
}

void doWork()
{
    analyze(globalThing);
}
```

Function `analyze` cannot throw an exception, since the **noexcept** specifer indicates this function would not throw an exception. However, the `doWork` function itself may throw an exception as it does not specify the **noexcept**.

6.6

6.6 When each of the programs listed below is executed, the **`throw`** statement marked by a comment of the form `/* throw site */` will be reached, resulting in an exception being thrown. Identify the objects that are destroyed during the stack unwinding process initiated by this **`throw`** statement and specify the sequence in which these objects are destroyed.

(a)

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

using namespace std::literals;

void func1() {
    std::string hello("hello"s);
    std::vector<int> countdown{3, 2, 1, 0};
    {
        std::string die("die"s);
        std::vector<std::string> die3{die, die, die};
        throw std::runtime_error("yikes!"); /* throw site */
    }
    std::string goodbye("goodbye"s);
}

void func2(bool flag) {
    std::vector<double> dv{1.0, 0.5, 0.25};
    std::string herb("Herb"s);
    if (flag) {
        std::string bjarne("Bjarne"s);
        std::complex<double> i(1.0i);
        func1();
    }
}

int main() {
    std::string scott("Scott"s);
    std::vector<int> count{1, 2, 3};
    try {
        std::complex<double> z(1.0i);
        std::complex<double> u(z * z);
        func2(true);
    }
    catch (std::runtime_error& e) {
        std::cout << e.what() << "\n";
    }
}
```

The objects are destroied in the order of `die3`, `die`, `countdown`, `hello`, `i`, `bjarne`, `herb`, `dv`, `u`, `z`.

(b)

```
1  #include <iostream>
2  #include <stdexcept>
3  #include <string>
4  #include <cmath>
5  #include <vector>
6
7  using namespace std::literals;
8
9  [[noreturn]] void panic(std::string s)
10 {
11    throw std::runtime_error(s); /* throw site */
12 }
13
14 double squareRoot(double x)
15 {
16    if (x < 0.0) {
17       std::string s("square root of negative number"s);
18       panic(s);
19    }
20    return std::sqrt(x);
21 }
22
23 int main()
24 {
25    std::vector<double> v{1.0, 4.0, -1.0};
26    for (auto x : v) {
27       try {
28          std::cout << squareRoot(x) << "\n";
29       }
30       catch (std::runtime_error& e) {
31          std::cout << "exception: " << e.what() << "\n";
32       }
33    }
34 }
```

When x = 1.0 and 4.0, no objects are destroied by the throw statement at line 11.

When x = -1.0, the objects are destroied in the order of `s`.

6.8

6.8 For each of the code listings given below, identify any exception-safety problems and suggest how they can be eliminated.

(a)

```
1  #include <cstddef>
2
3  void useBuffers(std::size_t, char* p1, char* p2) noexcept;
4
5  void func(std::size_t size)
6  {
7     // allocate memory for first buffer
8     char* buf1 = new char[size];
9     // allocate memory for second buffer
10    char* buf2 = new char[size];
11    // use the buffers
12    useBuffers(size, buf1, buf2);
13    // deallocate memory for first buffer
14    delete[] buf1;
15    // deallocate memory for second buffer
16    delete[] buf2;
17 }
```

The buf1 and buf2 may fail to be allocated memory and cause memory leak. Use smart pointer for the memory allocation to eliminate the problem. I am not sure if the code inside `useBuffers` function will guarantee not to throw exception.

(b)

```cpp
#include <iostream>
#include <ios>
#include <iomanip>

// print int to stream in hexadecimal
bool print(std::ostream& out, int x)
{
    // save formatting flags
    auto oldFlags = out.flags();
    // set formatting flags to use hexadecimal and output integer value
    out << std::showbase << std::hex << x << "\n";
    // restore formatting flags
    out.flags(oldFlags);
    return out;
}
```

There is probably an exception thrown when output x, the oldFlags may not be restored. Use RAII object to save and restore the state.

(c)

```cpp
#include <deque>

// Note:
// std::deque::front: guaranteed not to throw
// std::deque::pop_front: guaranteed not to throw
// std::deque::push_back: may throw, provides strong guarantee

// FIFO queue class
template <class T>
class Queue
{
public:
    // get number of elements on queue
    int size() const {
        return q_.size();
    }
    // remove and return element from front of queue
    T get() {
        T result(q_.front());
        q_.pop_front();
        return result;
    }
    // add element to back of queue
    void put(const T& value) {
        q_.push_back(value);
    }
private:
    // underlying queue
    std::deque<T> q_;
};
```

Although, the `push_back()` function provides strong guarantee, it sitll possibly throw exception. Use function try block to handle the exception.