Exam, FE5226
(2018/2019, Semester 1)

Question 1 [10 marks]

What do each of the following small programs print?

Part A

```
#include <iostream>
using namespace std;

struct A { void foo() { cout << "Hello from A\n"; } };
struct B : A { void foo() { cout << "Hello from B\n"; } };

int main() {
    B b{};
    A& a = b;
    a.foo();
    return 0;
}
```

Part B

```
#include <iostream>
using namespace std;

struct A { virtual void foo() { cout << "Hello from A\n"; } };
struct B : A { virtual void foo() { cout << "Hello from B\n"; } };

int main() {
    B b{};
    A& a = b;
    a.foo();
    return 0;
}
```

Question 2 [15 marks]

Each of the following code snippets contains one error which will cause problems at runtime. Assume that all necessary STL header files are duly included. Identify the errors and explain the type of problem. How can the error be fixed?

Part A

```
struct ITrade {};
struct Swap : ITrade {
    Swap(int *tenorbegin, int *tenorend)
        : m_tenors(tenorbegin, tenorend)
    {}
    std::vector<int> m_tenors;
};

void foo(int *tenorbegin, int *tenorend)
{
    ITrade *p = new Swap(tenorbegin, tenorend);
    delete p;
}
```

Part B

```
size_t foo(size_t x)
{
    size_t n;
    std::cin >> n; // assume the user enter a valid number
    size_t *buffer = new size_t[n];
    for (size_t i = 0; i < n; ++i)
        buffer[i] = std::rand() % 100;
    size_t result = buffer[rand() % n];
    if (result < x)
        throw 0;
    delete [] buffer;
    return result;
}
```

Part C

```
int foo()
{
    const int values[] = { 51, -4, 23, 50 };
    return values[rand() % 5];
}
```

Question 3 [20 marks]

The function *my_lower_bound* implemented below performs a binary search (at every iteration the size of the search range is roughly divided by two) and returns the index of the first element in the sorted array x that is not less than v. If no such element is found, then the function returns n. This is similar to the STL *lower_bound* function, except that instead returning an iterator, it returns an array index.

```
size_t my_lower_bound
    ( const int *x // pointer to first element in the sorted range
    , size_t n  // number of elements in the range
    , int v  // sought value
    )
{
    size_t first = 0;
    while (n > 0) {
        size_t n_half = n / 2;
        size_t mid = first + n_half;
        if (x[mid] < v) {
            first = mid + 1;
            n -= n_half;
        }
        else
            n = n_half;
    }
    return first;
}
```

Let *int* $x = \{1, 3, 5\}$, we expect that the following expressions should be *true*:
1) my_lower_bound(x, 3, 0) == 0
2) my_lower_bound(x, 3, 1) == 0
3) my_lower_bound(x, 3, 2) == 1
4) my_lower_bound(x, 3, 3) == 1
5) my_lower_bound(x, 3, 4) == 2
6) my_lower_bound(x, 3, 5) == 2
7) my_lower_bound(x, 3, 6) == 3
Unfortunately there is a bug in the function. Because of that, some (or all) of the above expressions either evaluate to *false* or cause runtime errors.
Part A Which of the above expressions either evaluate to *false* or cause runtime errors?
Part B What is the bug and how can it be fixed?
Part C Trace the variables *first*, *n_half*, *n* and *mid* for expression number (4) after the bug has been fixed.

Question 4 [15 marks]

What is the output of the program listed below?

```
#include <iostream>
#include <utility>

std::pair<int, int> foo_helper(unsigned n)
{
    if(n > 1) {
        auto p = foo_helper(n-1);
        return std::make_pair(p.second, p.first + p.second);
    }
    if(n==1)
        return std::make_pair(0,1);
    return std::make_pair(0,0);
}

unsigned foo(unsigned n)
{
    return foo_helper(n).second;
}

int main()
{
    for (unsigned i = 0; i < 10; ++i)
        std::cout << foo(i) << ", ";
    std::cout << "\n";
    return 0;
}
```

Question 5 [20 marks]

The user guide of the STL function *sort* is given in appendix A. What is the output of the program listed below?

```
#include <iostream>
#include <algorithm>
#include <array>
using namespace std;

void printArray(unsigned arrayNo, const array<int, 5>& x){
    cout << "array " << arrayNo << ": ";
    for (auto xi: x) cout << xi << ", ";
    cout << "\n";
}

struct lessabs { bool operator()(int a,int b) { return abs(a) < abs(b); } };

int main() {
    array<int, 5> z = { -5, 2, 7, -4, 0 }, x;

    printArray(0, z);

    x = z;
    sort(x.begin(), x.end());
    printArray(1, x);

    x = z;
    sort(x.begin(), x.end(), greater<int>{});
    printArray(2, x);

    x = z;
    sort(x.begin(), x.end(), lessabs{} );
    printArray(3, x);

    x = z;
    array<int, 5> index = { 0, 1, 2, 3, 4};
    sort(index.begin(), index.end(),
         [&x](int a, int b) { return x[a] < x[b]; } );
    printArray(4, index);

    return 0;
}
```

Question 6 [10 marks]

Part A

Implement a function with function head
double cnorm(**double**);
which computes the normal cumulative standard distribution $N(x)$ using the complementary error function $erfc(x)$ given in the STL. The relation between the two is:

$$N(x) = \frac{1}{2} erfc\left(-\frac{x}{\sqrt{2}}\right)$$

Part B

Implement a function with function head
double black(**double** f, **double** k, **double** v, **double** t);
which computes the un-discounted price of a call option on a stock with strike k expiring at time t, when the forward price of the stock for time t is f and its volatility is v. The formula is:

$$price(x) = f\, N(d1) - k\, N(d2)$$

where

$$d1 = \frac{\ln \frac{f}{k}}{\sqrt{v^2 t}} + \frac{1}{2}\sqrt{v^2 t}$$
$$d2 = d1 - \sqrt{v^2 t}$$

Avoid duplicate calculations in the function.

Question 7 [10 marks]

Each of the following code snippets contains one error which will cause compilation to fail. Assume that all necessary STL header files are duly included and that we specified **using namespace** std. Identify the errors and explain the type of problem. How can the error be fixed?

Part A

```
int sum(array<int,4>& a)
{
    int s = 0;
    for (auto i : a)
        s += i;
    return s;
}
int main()
{
    const array<int,4> x = {1, 2, 3, 4};
    cout << sum(x) << "\n";
    return 0;
}
```

Part B

```
struct A
{
    A(int x) : m_x(x) {}
    int m_x;
};

struct B
{
    B(int x) : A(x) {}
};

int main()
{
    B b(2);
    cout << b.m_x << "\n";
    return 0;
}
```

END OF PAPER

1 Solution

Question 1 [10 marks]

Part A

```
Hello from A
```

Part B

```
Hello from B
```

Question 2 [15 marks]

Part A

The ITrade destructor is not declared as *virtual*. Therefore, when the object pointed by the ITrade pointer is deleted, the destructor of the Swap class is not called and the vector m_tenor is not be destroyed. So there is a memory leak. To fix the problem, the destructor of ITrade must be declared as virtual.

```
struct ITrade { ~ITrade() {} };
```

Part B

The function has two exit points, one at the end and one if the exception is thrown. In the latter case the memory block pointed by *buffer* is not deallocated, i.e. there is a memory leak. To fix the problem, either a specific deallocation is added before the exception is thrown (see listing 1), or the raw pointer is pointer is replaced by a smart pointer (see listing 2), in which case the delete instruction is no longer necessary, or move deallocation before the if-statement (see listing 3).

Listing 1: specifically deallocate

```
...
if (result < x) {
    delete [] buffer;
    throw 0;
}
...
```

Listing 2: use smart pointers

```
...
std::unique_ptr<size_t[]> buffer = new size_t[n];
...
if (result < x)
    throw 0;
return result;
```

Listing 3: deallocate before if-statement

```
...
delete [] buffer;
if (result < x)
    throw 0;
return result;
```

Part C

The instruction (rand() % 5) can generate a value in the range $[0, 4]$. The value 4 is an invalid index for the array values, which contains only 4 elements, and would cause an invalid memory access. To fix it, the index should not exceed the value 3, i.e. it should be:

```
...
return values[rand() % 4];
```

Question 3 [20 marks]

Part A

Expressions failing are 5, 6, and 7, i.e.
5) my_lower_bound(x, 3, 4) == 2
6) my_lower_bound(x, 3, 5) == 2
7) my_lower_bound(x, 3, 6) == 3
which cause runtime error

Part B

For some values of the inputs, as the algorithm progress, the index *mid* can become too large for the array x. This is because when $(x[mid] < v)$ is *true* the size of the interval n should decrease by *n_half+1*, not by *n_half*. Hence the fix is:

```
...
if (x[mid] < v) {
    first = mid + 1;
    n -= n_half + 1;  // fixed
}
else
    n = n_half;
...
```

Part C

first	nhalf	n	mid
		3	
0			
	1		
			1
		1	
	0		
			0
1			
		0	

Question 4 [15 marks]

```
0, 1, 1, 2, 3, 5, 8, 13, 21, 34,
```

Question 5 [20 marks]

```
array 0: -5, 2, 7, -4, 0,
array 1: -5, -4, 0, 2, 7,
array 2: 7, 2, 0, -4, -5,
array 3: 0, 2, -4, -5, 7,
array 4: 0, 3, 4, 1, 2,
```

Question 6 [10 marks]

Part A

```
double cnorm(double x)
{
    return 0.5 * erfc(x / (-sqrt(2)));
}
```

Part B

```
double black(double f, double k, double v, double t)
{
    double stdev = v * sqrt(t);
    double d1 = log(f / k) / stdev + 0.5 * stdev;
    double d2 = d1 - stdev;
    return f * cnorm(nd1) - k * cnorm(nd2);
}
```

Question 7 [10 marks]

Part A

The array x is declared as *const*, hence it cannot be passed by reference to the function *sum*, which takes as argument a non-*const* array reference. Since the function *sum* does not modify the array, the obvious fix is to declare its argument as *const*.

```
int sum(const array<int,4>& a)
...
```

Part B

The constructor of B invokes the constructor of A in its initializer list, but A is not a base class of B. The constructor of B used suggests that it was programmer intention for B to inherits from A, hence that is the fix.

```
...
struct B : A  // B inherits from A
...
```

Appendix A

function template

std::sort <algorithm>

```
default (1)  template <class RandomAccessIterator>
               void sort (RandomAccessIterator first, RandomAccessIterator last);
custom (2)   template <class RandomAccessIterator, class Compare>
               void sort (RandomAccessIterator first, RandomAccessIterator last, Compare comp);
```

Sort elements in range

Sorts the elements in the range `[first,last)` into ascending order.

The elements are compared using `operator<` for the first version, and *comp* for the second.

Equivalent elements are not guaranteed to keep their original relative order (see `stable_sort`).

Parameters

`first, last`

Random-access iterators to the initial and final positions of the sequence to be sorted. The range used is `[first,last)`, which contains all the elements between *first* and *last*, including the element pointed by *first* but not the element pointed by *last*.
`RandomAccessIterator` shall point to a type for which `swap` is properly defined and which is both *move-constructible* and *move-assignable*.

`comp`

Binary function that accepts two elements in the range as arguments, and returns a value convertible to `bool`. The value returned indicates whether the element passed as first argument is considered to go before the second in the specific *strict weak ordering* it defines.
The function shall not modify any of its arguments.
This can either be a function pointer or a function object.

Example

```
// sort algorithm example
#include <iostream>      // std::cout
#include <algorithm>     // std::sort
#include <vector>        // std::vector

bool myfunction (int i,int j) { return (i<j); }

struct myclass {
  bool operator() (int i,int j) { return (i<j);}
} myobject;

int main () {
  int myints[] = {32,71,12,45,26,80,53,33};
  std::vector<int> myvector (myints, myints+8);               // 32 71 12 45 26 80 53 33

  // using default comparison (operator <):
  std::sort (myvector.begin(), myvector.begin()+4);           //(12 32 45 71)26 80 53 33

  // using function as comp
  std::sort (myvector.begin()+4, myvector.end(), myfunction); // 12 32 45 71(26 33 53 80)

  // using object as comp
  std::sort (myvector.begin(), myvector.end(), myobject);     //(12 26 32 33 45 53 71 80)

  // print out content:
  std::cout << "myvector contains:";
  for (std::vector<int>::iterator it=myvector.begin(); it!=myvector.end(); ++it)
    std::cout << ' ' << *it;
  std::cout << '\n';

  return 0;
}
```

Output:

```
myvector contains: 12 26 32 33 45 53 71 80
```

(source: http://www.cplusplus.com/reference/algorithm/sort/)