Datum: Jméno a Příjmení:.....

1. Uvažujte následující program. Co bude na standardním výstupu po jeho provedení? [4B]

```
class object {
public:
    object() {
        std::cout << "1\n";
    }
    object(const object&) {
        std::cout << "2\n";
    }
    object& operator=(const object&) {
        std::cout << "3\n";
        return *this;
    }
    object(object&&) {
        std::cout << "4\n";
    }
    object& operator=(object&&) {
        std::cout << "5\n";
        return *this;
    }
    ~object() {
        std::cout << "6\n";
    }
};

void foo(object x) {
    std::cout << "foo!\n";
}

int main() {
    object o1;
    object o2;
    object o3 = std::move(o2);
    foo(o3);
    o1 = o2;
}
```

Řešení:

1
1
4
2
foo!
6
3
6
6
6

2. Obsahuje konstruktor třídy `array` chybu? Pokud ano, jakou? [2B]

```
template <typename T>
class array {
public:
    array(T* data, std::size_t length) :
        elements{ new T[len] }, len{ length } {
        for (int i = 0; i < len; ++i) {
            elements[i] = data[i];
        }
    }
```

```
private:
    T* elements;
    std::size_t len;
};
```

Proměnná len je použita při alokaci pole T, před svou inicializací.

3. Jaká hodnota se vytiskne na výstup při provedení tohoto fragmentu kódu? [3B]

```
void foo() {
    std::vector<int> vec = { 1, 2, 3, 2, 3, 7, 7, 5, 2 };
    int i = 0;
    std::for_each(begin(vec), end(vec), [i](int elem) mutable {
        static int temp = 0;
        temp += elem % 3;
        i += temp + elem % 4;
    });
    std::cout << i << std::endl;
}
```

0, protože lambda bere proměnnou i hodnotou a tudíž se změny i neprojeví mimo lambdu.

4. Navrhněte interface šablonovaného zásobníku, `stack<T>`. O T předpokládejte, že podporuje kopírující operace. Následující program předvádí, jak má zásobník fungovat. [5B]:

```
int main() {
    stack<int> s1;
    stack<int> s2;

    s1.push(12);
    s1.push(13);
    s1.push(17);

    s2 = s1;
    while (!s2.empty()) {
        int temp = s2.top();
        s2.pop();
        std::cout << temp << std::endl;
    }
    std::cout << std::boolalpha << s1.empty() << std::endl;
}
```

```
Očekávaný výstup je
17
13
12
False
```

```
template <typename T>
class stack {
public:
    stack();
    stack(const stack<T>& rhs);
    stack& operator=(const stack<T>& rhs);
    void push(const T&);
    void pop(); // Bereme i T pop(); ale v praxi silně nedoporučujeme
    T& top();
    bool empty() const;
};
```

5. Naimplementujte kopírující konstruktor pro tuto třídu: [6B]:

```
class type {
public:
    ~type() {
        delete pi;
    }
private:
    int i;
    const int ci;
    int& ri;
    int* pi;
    type* parent;
};

type::type(const type& rhs):
          i(rhs.i), ci(rhs.ci),
          ri(rhs.ri), pi(new int), parent(parent) {
    *pi = *rhs.pi;
}
```