

Object oriented programming – Elektrijska 2022 (Sunny Beach)

Task 1 (14):

Task 2 (14):

Task 3 (14):

T.1. (14)	T.2. (14)	T.3. (14)	T.4. (14)	T.5. (14)	T.6. (14)	T.7. (16)	Total

Task 4 (14):
Task 5 (14):
Task 6 (14):
Task 7 (16):

Additional space:

Object oriented programming – Elektrijska 2022 (Sunny Beach)

Task 1. What is the output for the following program?

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

class Base {
protected:
    int x;
public:
    Base(int a) : x(a) {};
    virtual int f(int);
    double f(double);
    virtual int g(int i = 10);
};

int Base::f(int a) { return a + x; }

double Base::f(double a) { cout << "Base::f(double)" << endl; return a + x / 2; }

int Base::g(int i) {
    if (i == 10) { x = f(i << 1); }
    cout << "Base::g(int) " << i << " " << x << endl;
    return x;
}

class Derived : public Base {
protected:
    int x;
public:
    Derived(int a, int b) : Base(a), x(b) {};
    virtual double f(Base a);
    int g(int i = 20);
};

double Derived::f(Base a) { cout << "Derived::f(Base)" << endl; return a.f(x) + x; }

int Derived::g(int i) {
    if (i == 10) { x = f(i << 1); }
    if (i == 20) { x = f(i << 1 | 1); }
    cout << "Derived::g(int) " << i << " " << x << endl;
    return x;
}

class DerivedSunny : public Derived {
public:
    DerivedSunny(int a, int b) : Derived(--a, ++b) {};
    int f(int);
    virtual double f(double);
};

int DerivedSunny::f(int a) { cout << "Sunny::f(int)" << endl; return a + x * 2; }

double DerivedSunny::f(double a) { cout << "Sunny::f(double)" << endl; return a + x / 4; }

class DerivedBeach : public DerivedSunny {
public:
    DerivedBeach(int a, int b) : DerivedSunny(++a, ++b) {};
    int g(int i = 40);
};
```

```

int DerivedBeach::g(int i) {
    if (i == 10) { x = f((i + x) << 1); }
    if (i == 20) { x = f((i + x) << 1 | 1); }
    if (i == 40) { x = f(((i + x) << 1 | 1) << 1); }
    cout << "Beach::g(int) " << i << " " << x << endl;
    return x;
}

class DerivedSea : public DerivedBeach {
public:
    DerivedSea(int a, int b) : DerivedBeach(++a, --b) {};
    int f(int);
    double f(double);
    double f(Base& a);
};

int DerivedSea::f(int a) { cout << "Sea::f(int)" << endl; return a + x * 2; }

double DerivedSea::f(double a) {
    cout << "Sea::f(double)" << endl; return a + x / 2;
}

double DerivedSea::f(Base& a) {
    cout << "Sea::f(Base)" << endl; return a.f(x) + x;
}

int main()
{
    Derived      d(1, 3);
    DerivedSunny ds(2, 4);
    Base* pbs = new DerivedSunny(2, 4);
    DerivedBeach db(2, 4);
    DerivedSunny* psb = new DerivedBeach(2, 4);
    Derived* pds = psb;
    DerivedSea dsea(2, 4);
    DerivedBeach* pbsea = new DerivedSea(2, 4);
    DerivedSunny* pssea = pbsea;
    Derived* pdsea = pbsea;

    pbs->f(1);
    pbs->f(2.0);

    d.f(2.0);
    d.g();

    pbsea->f(2.0);
    psb->g();

    dsea.f(db);
    pdsea->f(db);

    pssea->f(2);
    pbsea->f(2);
    pbsea->g();

    return 0;
}

```

Task 2. What is the output for the following program?

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

class A {
public:
    static int id;
    A() : objID(id) {
        cout << ++id << " cons A" << endl;
    }
    A(int objectNumber) {
        id += objID = objectNumber;
        cout << "cons A (" << id << ")" << endl;
    }
    ~A() {
        cout << --id << " dest " << endl;
    }
    virtual void f() {
        cout << "A::f -> " << objID << endl;
    }
protected:
    int objID;
};
int A::id = 0;

class B : virtual public A {
public:
    B() : A(id) {
        cout << (id*=2) << " cons B" << endl;
    }
    B(int i) : A(i) { cout << "cons B (" << id << ")" << endl; }
};

class C : virtual public A {
    A a;
public:
    C() : A(id) {
        cout << (id <= 1) << " cons C" << endl;
        A b(id);
    }
    C(int i) : A(i) { cout << "cons C (" << id << ")" << endl; }
};

class D : public B, virtual public C {
public:
    D(int i) : C(id) {
        cout << "cons D (" << id << ")" << endl;
    }
};
```

```

class E : public B, public D {
public:
    E(int i) : B(i), D(i) {
        static B b(i);
        cout << "cons E (" << i << ")" << endl;
    }
    E(const E& e) : D(e) {
        cout << "cons E (" << i << ")" << endl;
    }
    virtual void f() {
        cout << "E::f -> " << objID << endl;
    }
};

```

```

int main() {
    E e(0);
    E ee = e;
    C& rc = ee;
    D *pd = &e;
    A* pa = &rc;
    rc.f();
    pd->f();
    pa->f();
    exit(0);
}

```

Task 3. What is the output for the following program?

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

template <int n>
class A {
protected:
    int X;
    A<n> *aa[n];
public:
    A<n>() : A<n>(n) { }
    A<n>(int x) : X(x) {
        for (int i = 0; i<n; i++)
            this->aa[i] = nullptr;
    }
    A<n>** operator[](int ind) {
        return &this->aa[ind];
    }
    friend ostream& operator<<(ostream& os, A& a) {
        cout << a.X * a.X << " ";
        return os;
    }
    virtual void drive() {
        cout << *this;
        for (int i=0; i<n; i++)
            if (this->aa[i] != nullptr)
                this->aa[i]->drive();
    }
};

template <int n>
class B : public A<n> {
public:
    B<n>(int x) :A<n>(x) { };
    void transform() {
        for (int i = 0; i < n; i++) {
            A<n>* t = this->aa[i];
            this->aa[i] = this->aa[n - i - 1];
            this->aa[n - i - 1] = t;
        }
    }
    void drive() {
        transform();
        for (int i = 0; i < n; i++) {
            if (this->aa[i] != nullptr)
                this->aa[i]->drive();
            if (i % 2 == 0) cout << *this;
        }
    }
};

template <int n>
class C : public A<n> {
public:
    C<n>(int x) : A<n>(x) {}
    void drive() {
        for (int i = 0; i < n; i++) {
            if (this->aa[i] != nullptr)
                this->aa[i]->drive();
        }
        cout << *this;
    }
};
```



```

template <int n>
class D : public A<n> {
    A<n> *a;
    int it;
public:
    D<n>(int x, A<n> *ab) : A<n>(x), a(ab), it() {}
    void transform() {
        A<n>* t = this->aa[0];
        for (int i = 1; i < n; i++)
            this->aa[i-1] = this->aa[i];
        this->aa[n - 1] = t;
    }
    void drive() {
        transform();
        for (int i = 0; i < n; i++) {
            if (this->aa[i] != nullptr)
                this->aa[i]->drive();
        }
        cout << *this;
        cout << "| ";
        if (it++ <= 0) a->drive();
        cout << "| ";
    }
};

int main() {
    const int size = 2;
    A<size>* a[12], *t = nullptr;
    for (int i = 0; i<10; i++) {
        switch (i % 4) {
            case(0):    a[++i] = new A<size>(i);    break;
            case(1):    a[++i] = new B<size>(i);    break;
            case(2):    a[++i] = new C<size>(i);    break;
            case(3):    a[++i] = new D<size>(i, a[i / 2]);    break;
            default:    a[++i] = nullptr;           break;
        }
        if (t == nullptr) {
            t = a[i--];
        } else {
            *((*t)[i%size]) = a[i];
            if (--i%size == 0)
                t = a[i / 2 + 1];
        }
    }
    a[1]->drive();
    return 0;
}

```

Task 4. What is the output for the following program?

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

class Basic {
protected:
    int data;
public:
    Basic() { data = 0; }
    Basic(int d) : data(d) {}
    Basic(const Basic& b) : data(b.data) { }
    Basic& operator=(const Basic &b) {
        data = b.data;
        return *this;
    }
    bool operator>(Basic &b) {
        return Value() > b.Value();
    }
    Basic operator+(const Basic& b) {
        return Basic(data + b.data);
    }
    Basic& operator+=(const Basic& b) {
        data += b.data;
        return *this;
    }
    virtual int Value() {
        return data;
    }
    virtual void Print() {
        cout << "(" << data << ") ";
    }
};

class Extended : public Basic {
protected:
    int weight;
public:
    Extended() { weight = 1; }
    Extended(int d, int w) : Basic(d), weight(w) {}
    Extended(const Extended& e) : Basic(e.data), weight(e.weight) { }
    Extended& operator=(const Extended &e) {
        data = e.data;
        weight = e.weight;
        return *this;
    }
    virtual int Value() {
        return weight * data;
    }
    virtual void Print() {
        cout << "(" << data << "," << weight << ") ";
    }
};

template <int dim>
class Container : public Basic {
    Basic *els[dim];
    int n;
public:
    Container<dim>() : n(0) {
        for (int i = 0; i<dim; i++)
            els[i] = NULL;
    }
    void Add(Basic *elem) {
        els[n++] = elem;
        data += elem->Value();
    }
};
```

```

const Basic& operator[](int ind) const {
    return *els[ind];
}
Basic& operator[](int ind) {
    return *els[ind];
}
Container& operator+=(const Basic& el) {
    Container& c = *this;
    for (int i = 0; i < n; i++)
        c[i] = c[i] + el;
    return c;
}
Container operator+(const Container& con) {
    Container cc(*this);
    Container& c = *this;
    for (int i = 0; i < n; i++)
        cc[i] = c[i] + con[i];
    return cc;
}
void Join(Container& con1, Container& con2, Container* con3) {
    Container& c = *this;
    for (int i = 0; i < n; i++)
        c[i] = con1[i] + con2[i] + con3[i];
}
void Merge(Container& con1, Container& con2, Container* con3) {
    Container& c = *this;
    for (int i = 0; i < n; i++)
        if (con1[i] > con2[i])
            c[i] += con1[i] > con3[i] ? con1[i] : con3[i];
        else
            c[i] += con2[i] > con3[i] ? con2[i] : con3[i];
}
void Print() {
    Basic::Print();
    for (int i = 0; i < n; i++)
        els[i]->Print();
    cout << endl;
}

};

int main() {
    const int size = 5;
    int data = 64, weight = 2;
    Container<size> con[size];
    for (int i = 0; i < size; i++) {
        if (i <= size / 2) {
            con[0].Add(new Basic(data -= data / 2));
            con[1].Add(new Basic(data%size));
            con[2].Add(new Basic());
            con[3].Add(new Basic());
            con[4].Add(new Basic());
        }
        else {
            con[0].Add(new Extended(data -= data / 2, weight *= 2));
            con[1].Add(new Extended(data%size, weight / 4));
            con[2].Add(new Extended());
            con[3].Add(new Extended());
            con[4].Add(new Extended());
        }
    }
    con[2] = con[0] + con[1];
    con[3].Join(con[2], con[1], &con[0]);
    con[4].Merge(con[2], con[1], &con[0]);
    for (int i = 0; i < size; i++)
        con[i].Print();
    return 0;
}

```

Task 5. What is the output for the following program?

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

template <class T, class S, int n>
class Safe
{
protected:
    int a;
public:
    Safe<T, S, n>(int aa = n)
    {
        a = aa;
    };
    float f(float f)
    {
        if (f < 0) throw f;
        return a + f;
    };
    double f(double& d)
    {
        if (d < a) throw d;
        return d - a;
    };
    virtual int f(const Safe<T, S, n>& s)
    {
        if (s < *this) throw this;
        return s - *this;
    };
    bool operator<(const Safe<T, S, n>& s) const
    {
        return a < s.a;
    }
    int operator-(const Safe<T, S, n>& s) const
    {
        return a - s.a;
    }
};

template <class T, int n>
class Confused : public Safe<T, char, n>
{
public:
    Confused<T, n>(int aa = n) : Safe<T, char, n>(aa) { };
    int f(const Safe<T, char, n>& s)
    {
        if (s < *this) throw Safe<T, char, n>::f(s);
        return s - *this;
    };
};

class Crazy : public Confused<int, -1>
{
public:
    Crazy(int aa = -1) : Confused<int, -1>(aa) { };
};
```

```

template <class T, class S, int n>
void process(Safe<T, S, n> &s, int i) {
    const int ci = -1;
    switch (i) {
    case 0: s.f(ci);
    case 1: s.f(Safe<int, char, -1>(ci - 1));
    case 2: s.f(ci - 1.0);
    case 5: s.f(Confused<int, -1>(1 - ci));
    case 6: s.f(Confused<int, -1>(ci - 1));
    }
}

template <class T, class S, int n>
void inspect(int i) {
    int ci = 1;
    switch (i) {
    case 3: throw ci;
    case 4: throw ci + 1.0;
    case 7: throw Safe<T, S, n>();
    case 8: throw Confused<T, n>();
    case 9: throw new Confused<T, n>();
    }
}

int main()
{
    Safe<int, char, -1> s;
    Crazy c;
    for (int i = 0; i<10; i++) {
        cout << i << ":";
        try {
            Safe<int, char, -1> &rs = i < 5 ? c : s;
            process<int, char, -1>(rs, i);
            try {
                inspect<int, char, -1>(i);
            }
            catch (Safe<int, char, -1> *sf) { cout << "Biti"; throw; }
            catch (float) { cout << "Corpore"; }
            catch (double) { cout << "Mens"; }
            catch (Safe<int, char, -1> st) { cout << "Bit"; throw; }
            catch (Confused<int, -1> &cf) { cout << "Meghan"; throw; }
            catch (Crazy &cy) { cout << "Sta"; throw; }
        }
        catch (Crazy *cy) { cout << "Cemo"; }
        catch (Confused<int, -1> *cf) { cout << "Zdrava"; }
        catch (Safe<int, char, -1> *st) { cout << "Mora"; }
        catch (float) { cout << "Corpore"; }
        catch (double) { cout << "Mens"; }
        catch (Safe<int, char, -1> sf) { cout << "BitBit"; }
        catch (Confused<int, -1> &cf) { cout << "Markle"; }
        catch (Crazy &cy) { cout << "Sad"; }
        catch (...) { cout << "Moze"; }
        cout << endl;
    }
    return 0;
}

```

Task 6. What is the output for the following program?

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

class Base {
protected:
    int x;
public:
    Base(int a = 1) : x(a) { cout << "Base(int) " << ++x << endl; };
    Base(const Base& b) : Base(b.x) { cout << "Base(Base) " << x << endl; };
    int f(int& n) { n += x >> 1; cout << "f = " << x << endl; return x = n; }
    virtual int g(int m) { m += x << 1; cout << "g = " << x << endl; return x
= m; }
};

class Derived : public Base {
public:
    Derived(int a) { g(a); };
    virtual int f(int& n) { int v = x + x&1; v += Base::f(n); return v; }
};

class Concrete : public Derived {
    Derived d;
    Base b;
public:
    Concrete(int a, const Base& bb, const Derived& dd) : Derived(a), b(bb),
d(dd) { b.g(0); };
    int f(int& n) { int v = d.f(n); v += Derived::f(n); return v; }
    virtual int g(int m) { int v = d.g(m); v += Derived::g(m); return v; }
    int f(int n, Base& bb) { int v = f(n); v += bb.f(n); return v; }
};

int main()
{
    int m = 3, n = 1;
    Base b;
    Derived d(1);
    Base* pbd = &d;
    b.f(n);
    pbd->f(n);
    d.g(m);
    pbd->g(m);

    Concrete c(1, b, d);
    Base* pbc = &c;
    Derived* pdc = &c;
    c.f(n, c);
    pbc->f(n);
    pdc->f(n);
    pdc->g(m);
    return 0;
}
```

Task 7. What is the output for the following program?

```
#include <iostream>
using namespace std;
#include <math.h>

template <class T>
class Elem
{
public:
    T val, key;
    Elem<T>() { val = T(); key = T(); }
    Elem<T>(T v, T k) { val = v; key = k; }
    bool operator<(const Elem<T>& t) {
        return val < t.val || val == t.val && key < t.key;
    }
    double merge(const Elem<T>& e1, const Elem<T>& e2) {
        return (e1.key - key) * (e2.val - val) - (e1.val - val) * (e2.key - key);
    }
    void print() {
        cout << "[" << key << ", " << val << "]" ";
    }
};

template <class T, int size>
class Collection
{
    Elem<T>* pairEls[2];
    int* lookup;
    int count;
public:
    Collection() { lookup = new int[2 * size](); count = 0; };
    void initialize(int arr[]) {
        pairEls[0] = new Elem<T>[size];
        for (int i = 0; i < size; i++) {
            pairEls[0][i] = Elem<T>((arr[i] & 0xF0) >> 0x4, arr[i] & 0xF);
        }
    }
    double sum(Elem<T>* els, int s, int e) {
        double acc = els[0].key * (els[s].val - els[e].val);
        acc += els[s].key * (els[s+1].val - els[0].val);
        for (int i = s+1; i < e; i++)
        {
            acc += els[i].key * (els[i + 1].val - els[i - 1].val);
        }
        acc += els[e].key * (els[0].val - els[e - 1].val);
        return fabs(acc / 2);
    }
    int search(int s, int e) {
        Elem<T>* els = pairEls[1];
        double p = sum(els, s, e);
        int l = s, r = e, m = -1;
        while (r - l > 1) {
            m = (l + r) / 2;
            if (2 * sum(els, s, m) < p)
                l = m;
            else
                r = m;
        }
        if (l != s && r != e)
            if (p - 2 * sum(els, s, l) < 2 * sum(els, s, r) - p)
                m = l;
            else
                m = r;
        return m;
    }
};
```

```

void create() { create(1, count - 1, 1); }
void create(int l, int r, int ind) {
    int m = search(l, r);
    lookup[ind] = m;
    if (l + 1 < m)
        create(l, m, 2 * ind);
    if (m < r - 1)
        create(m, r, 2 * ind + 1);
}
void sort(Elem<T>* els) {
    for (int i0 = 0; i0 < size-1; i0++)
        for (int iI = size - 1; iI > i0; iI--)
            if (els[0].merge(els[iI], els[iI - 1]) > 0) {
                Elem<T> t = els[iI - 1];
                els[iI - 1] = els[iI];
                els[iI] = t;
            }
}
void process() {
    Elem<T>* els = pairEls[0];
    int ex = 0;
    for (int i = 1; i < size; i++)
        if (els[i] < els[ex])
            ex = i;
    Elem<T> el = els[0];
    els[0] = els[ex];
    els[ex] = el;
    sort(els);
    pairEls[1] = new Elem<T>[size];
    Elem<T>* elsSel = pairEls[1];
    elsSel[count++] = els[0];
    elsSel[count++] = els[1];
    elsSel[count++] = els[2];
    for (int i = count; i < size; i++) {
        while (elsSel[count - 2].merge(elsSel[count - 1], els[i]) < 0) {
            count--;
        }
        elsSel[count++] = els[i];
    }
}
void print() {
    for (int i = 0; i < count; i++)
        pairEls[1][i].print();
    cout << endl;
}
void print(int ind) {
    if (lookup[ind] != 0) {
        pairEls[1][lookup[ind]].print();
        print(2 * ind);
        print(2 * ind + 1);
    }
}

};

int main() {
    const int nEls = 15;
    int arEls[] = { 0x32, 0x73, 0x87, 0xA8, 0x5A, 0x48,
        0x71, 0x8A, 0x69, 0x44, 0x25, 0xB5, 0xA2, 0x56, 0x39};
    Collection<double, nEls> collection;
    collection.initialize(arEls);
    collection.process();
    collection.print();
    collection.create();
    collection.print(1);
    return 0;
}

```


Answers: Object oriented programming – 2022 (Sunny Beach)

Task1.

```
Sunny::f(int)
Base::f(double)
Derived::f(Base)
Derived::f(Base)
Derived::g(int) 20 47
Sea::f(double)
Sunny::f(int)
Beach::g(int) 20 65
Sea::f(Base)
Sunny::f(int)
Derived::f(Base)
Sea::f(int)
Sea::f(int)
Sea::f(int)
Beach::g(int) 40 192
```

Task2.

```
1 cons A
2 cons A
4 cons C
cons A (8)
7 dest
cons B (7)
14 cons B
cons D (14)
cons A (14)
cons B (14)
cons E (14)
15 cons A
16 cons A
32 cons C
cons A (64)
63 dest
126 cons B
cons E (126)
E::f -> 14
E::f -> 0
E::f -> 14
125 dest
```

Task3.

```
0 64 49 | 25 36 4 | 9 | 0 49 | | 64 9
| | 1 16 81 25 36 4 | 1 16 81 25 36 4
```

Task4.

```
(88) (34) (17) (11) (8,4) (4,8)
(14) (2) (1) (3) (4,1) (2,2)
(88) (34) (17) (11) (8,4) (4,8)
(0) (124) (32) (102) (12,1) (6,1)
(0) (88) (17) (88) (8,1) (4,1)
```

Task5.

```
0:Corpore
1:Mora
2:Corpore
3:Moze
4:Mens
5:Mora
6:Mora
7:BitBitBit
8:BitBitBit
9:BitiZdrava
```

Task6.

```
Base(int) 2
Base(int) 2
g = 2
f = 2
f = 5
g = 4
g = 11
Base(int) 2
g = 2
Base(int) 26
Base(Base) 26
Base(int) 3
Base(Base) 3
g = 3
f = 26
f = 5
f = 19
f = 28
f = 17
f = 18
g = 26
g = 35
```

Task7.

```
[5, 2] [9, 3] [10, 5] [10, 8] [8, 10]
[5, 11] [2, 10] [1, 7] [2, 3]
[5, 11] [8, 10] [10, 8] [10, 5] [2,
10] [1, 7]
```