

## Seminar 6

Pentru fiecare dintre secvențele de cod de mai jos, spuneti dacă secvența compilează sau nu. În caz afirmativ, spuneti ce va afișa (în cazul în care standardul C++ spune că e comportament nedefinit, puteți specifica acest lucru). În caz negativ, sugerați o modificare, prin editarea a cel mult o linie de cod (modificarea unei linii de cod, adăugarea unei linii de cod sau ștergerea unei linii de cod), care să facă secvența să compileze și spuneti ce afișează noua secvență de cod.

### Secvența 1

```
1 #include<iostream>
2 struct Integer {
3     int x;
4     Integer(const int val = 0) : x(val){}
5     friend Integer operator+ (Integer& i, Integer& j) {
6         return Integer(j.x + i.x);
7     }
8     friend std::ostream& operator<<(std::ostream& o, Integer i) {
9         o << i.x; return o;
10    }
11 };
12 int main () {
13     Integer i(25), j(5), k(2020);
14     std::cout << (i + j + k);
15 }
```

## Secventa 2

```
1 #include <iostream>
2 #include <string>
3 struct A {
4     A (int i=0) {std::cout<<"A" << i;}
5     ~A () {std::cout<<"~A";}
6 };
7 struct B : public A {
8     A a;
9     B() : a(25) {std::cout<<"B";}
10    virtual ~B() {std::cout<<"~B";}
11 };
12 struct C: public B {
13     C () {std::cout << "C";}
14     ~C () {std::cout << "~C";}
15 };
16
17 int main () {
18     A *a = new C();
19     delete a;
20 }
```

## Secventa 3

```
1 #include <iostream>
2 struct A {
3     virtual void foo () {}
4 };
5 struct B : public A {
6     void foo () {}
7 };
8 class D: public B {
9 public:
10     void foo () {}
11     std::string bar() {return "bar";}
12 };
13 int main () {
14     A *p = new D();
15     if(dynamic_cast<B*>(p)) {std::cout << "Type is B";}
16     else if (dynamic_cast<D*>(p)) {std::cout << "Type is D";}
17     else { std::cout << "Conversion failed";}
18 }
```

## Secventa 4

```
1 #include <iostream>
2 struct A {
3     virtual void foo () {std::cout<<"A";}
4 };
5 struct B : public A {
6     void foo () {std::cout<<"B";}
7 };
8 class D: public B {
9 public:
10     void foo () {std::cout<<"D";}
11 };
12 int main () {
13     A *pa[] = {new B, new D, new B, new D, new B, new D};
14     for (int i = 0; i < 6; i++) {
15         if (i % 2 == 0) {
16             D d = dynamic_cast<D*>(*pa[i]);
17             d.foo();
18         } else {
19             B b = dynamic_cast<B*>(*pa[i]);
20             b.foo();
21         }
22     }
23 }
```

## Secventa 5

```
1 #include <iostream>
2 class C {
3 protected:
4     int x;
5 public:
6     C(int y): x(y) {}
7     virtual C operator+(const C& c) const {
8         return C(this->x + c.x);
9     }
10    friend std::ostream& operator << (std::ostream& o, C c) {
11        o << c.x; return o;
12    }
13 };
14
15 class D: public C {
16 public:
17     D(int y) : C(y) {}
18     C operator+(const C& c) const {
19         return C(x + 22);
20     }
21 };
22
23 int main () {
24     C *c = new D(4);
25     std::cout << *c + C(2);
26 }
```

## Secventa 6

```
1 #include <iostream>
2
3 int foo (int x, int y = 0) {
4     return x + y - 2020;
5 }
6
7 int foo (int x) {
8     return x + 2020;
9 }
10
11 int main () {
12     std::cout << foo(5);
13 }
```

## Secventa 7

```
1 #include <iostream>
2 struct A {
3     A (int i=0) {std::cout<<"A" << i;}
4     ~A () {std::cout<<"~A";}
5 };
6 struct B : public A {
7     A a;
8     B() : a(25) {std::cout<<"B";}
9     ~B() {std::cout<<"~B";}
10 };
11 struct C: public A, public B {
12     C () {std::cout << "C";}
13     ~C () {std::cout << "~C";}
14 };
15
16 int main () {
17     C c;
18 }
```

## Secventa 8

```
1 #include <iostream>
2 #include <string>
3 struct A {
4     A () {std::cout<<"A";}
5     ~A () {std::cout<<"~A";}
6 };
7 struct B : public A {
8     A a;
9     B() {std::cout<<"B";}
10    ~B() {std::cout<<"~B";}
11 };
12 struct C: public A, public B {
13     C () {std::cout << "C";}
14     ~C () {std::cout << "~C";}
15 };
16
17 int main () {
18     A *c = new C();
19 }
```



## Secventa 9

```
1 #include<iostream>
2
3 struct Integer {
4     int x;
5     Integer(const int val = 0) : x(val){}
6     Integer operator+ (Integer& i) {
7         return Integer(x + i.x);
8     }
9     friend std::ostream& operator<<(std::ostream& o, Integer& i) {
10         o << i.x; return o;
11     }
12 };
13
14 int main () {
15     Integer i(25), j(5), k(2020);
16     std::cout << (i + j + k);
17 }
```

## Secventa 10

```
1 #include <iostream>
2 struct A {
3     virtual void foo () {}
4 };
5 struct B : public A {
6     void foo () {}
7 };
8 struct D: public B {
9     void foo () {}
10    std::string bar() {return "bar";}
11 };
12 int main () {
13     A *p = new B();
14     D *pd = dynamic_cast<D*>(p);
15     if (pd != nullptr) { std::cout << "D"; }
16     else { std::cout << "incompatible"; }
17     std::cout<<pd->bar();
18 }
```

## Secventa 11

```
.
1 #include<iostream>
2 #include<climits>
3 struct Array {
4     int *x, size;
5     Array(const int& val = 5) : x(new int[val]), size(val){
6         for (int i = 0; i < size; i++) *(this->x+i) = i;
7     }
8     int operator[](unsigned i = 0) const {
9         return i > size ? INT_MAX : x[i];
10    }
11 };
12 int main () {
13     Array a(2020);
14     for (unsigned i = 0; i < a.size; i++) {
15         std::cout << a[i] << " ";
16     }
17 }
```

## Secventa 12

```
1 #include<iostream>
2 class B {
3 protected:
4     int x;
5 public:
6     B (int y = 2020) : x(y) {}
7 };
8
9 class D : public B {
10 public:
11     D(int y) : B(y) {}
12     D operator+ (const B& b) {
13         return D(x + b.x);
14     }
15     operator int () const {return x;}
16 };
17
18 int main () {
19     D d = (D(22) + D(5));
20     std::cout << d;
21 }
```

## Secventa 13

```
1 #include<iostream>
2 struct B {
3     virtual void foo () {std::cout<<"B";}
4 };
5 struct C: public B {
6     void foo () {std::cout<<"C";}
7 };
8 struct D: private B {
9     void foo () {std::cout<<"D";}
10 };
11 void call (B& b) {
12     b.foo();
13 }
14 int main () {
15     C c; D d;
16     call(c); call(d);
17 }
```