Авторское оформление кода сохранено. Выявленные во время написания коллоквиума студентами опечатки исправлены.

1. Если есть ошибки в операторах функции `main()`, исправьте их с помощью операции указания области видимости «::». Вычеркните операторы, которые не удалось исправить с помощью «::». Для оставшихся операторов с помощью операции «::» обозначьте ту область видимости, к которой относится вызываемый метод.

```
int x = 4;
class A { int x; public: A (int n = 1);
          virtual int f (int a = 0, int b = 0);  };
class B { int x; public: B (int n = 2);
                   int f (int a = 0); };
class C:  public A, public B { int x; public: C (int n = 3);
                   int f (int a, int b = 0);
                   int g (A * p);
};
A * p; C c;
int main() { p = & c;
             x = c.g (p);
             x = c.f ();
             x = c.f (x);
             x = c.f (x, 1);
             x = p -> f ();
             x = p -> f (x);
             x = p -> f (x, 1);  return 3; }
```

2. Укажите лишние и ошибочные операции динамического приведения типа, если таковые имеются в функции `main()`. Дайте необходимые пояснения своим исправлениям.

```
class K           { public:          void g () { cout << "K::g"; } };
class L: public K { public:          void f () { cout << "L::f"; } };
class M: public K { public: virtual  void h () { cout << "M::h"; } };
class P: public L { public:          void f () { cout << "P::f"; } };
class Q: public M { public: virtual  void h () { cout << "Q::h"; } };
class R: public P { public: virtual  void f () { cout << "R::f"; }
                            virtual  void h () { cout << "R::h"; } };
class S: public Q { public: virtual  void f () { cout << "S::f"; }
                            virtual  void h () { cout << "S::h"; } };
int main() {
        S os, * s = & os; K * k; L * l; M * m; P * p; Q * q; R * r;
    k = dynamic_cast <K *>(s);    s = dynamic_cast <S *>(k);
    l = dynamic_cast <L *>(k);    m = dynamic_cast <M *>(s);
    p = dynamic_cast <P *>(l);    q = dynamic_cast <Q *>(m);
    r = dynamic_cast <R *>(q);    s = dynamic_cast <S *>(p);    return 0; }
```

3. Для каждого вызова перегруженной функции с одним параметром укажите шаг алгоритма, на котором будет выбрана наиболее подходящая функция или выявлена неоднозначность. Для выбранной функции укажите её прототип.

```
int f(int a = 1){return a;}
int f(long double a = 5.0){return a;}
int main () {
 short int s; int i; bool b; float f = 1.0f; double d = 2.0;
      f (s);
      f (i);
      f (b);
      f (f);
      f (d);  }
```

4. Для объектов из задания 1 определить, какие конструкторы и деструкторы и в каком порядке будут выполняться при работе следующего фрагмента программы:

```
int main () { C c; class D { C c; B b; public: D (): b (c) { } } d; }
```

5. В открытой части класса `cl`, предназначенного для работы со сложными объектами, имеются конструктор и функция преобразования. Для облегчения работы с объектами этого класса создаётся дополнительный класс `ObjPtr`, в закрытую часть которого включается поле указателя на класс `cl`.

```
template <class T> class cl { /* ... */
   public: cl (const T s = 0);
           operator T ();
};
template <class T> class ObjPtr { /* ... */ cl<T> * c; public: /* ... */
};
```

Для класса «`ObjPtr`» написать реализацию операции изменения знака `-` и операции присваивания `=` таким образом, чтобы правильными оказались выражения (обязательно использовать и методы, и внешние функции):

```
ObjPtr <int> t1, t2;
/* ... */ t1 = 5; t2 = -t1; /* ... */
```

6. Что такое инкапсуляция и для чего она используется в Си++? Привести синтаксическую запись фрагмента программы, на котором можно проиллюстрировать понятие инкапсуляции.

7. Что будет выдано в стандартный поток вывода при работе следующей программы?

```
void f(X & x, int n);
struct X { X ()    { try  { f(*this, 1);  cout << "a"; }
                     catch (X){ cout << "b"; }
                     catch (int){ cout << "c"; }
                   }
           X (X &) { cout << "d"; }
  virtual ~X () { cout << "e"; }
};
struct Y: X { Y () { try { f (*this, 1);     cout << "f"; }
                     catch (Y){ cout << "g"; }
                     catch (int){ cout << "h"; }
                     cout << "i"; }
              Y (Y &){ cout << "j"; }
             ~Y (){ cout << "k"; }
};
void f(X & x, int n) { try  { if (n < 0) throw -n;
                              else if (n == 0) throw x;
                              else throw n; }
                       catch (int){ cout << "l"; } }
int main() { try { Y a; }
             catch (...){ cout << "m"; return 1; }
             cout << "n"; return 0; }
```

8. Если есть ошибки в следующем фрагменте, то в чём они заключаются? Исправьте ошибки, ничего не удаляя, добавив в общей сложности не более 29 символов.

```
class W         { public: int * w; void wf (int * wp) { w = wp; }};
class X: W      { public: int * x; void xf (int * xp) { x = xp; w = xp; }};
class Y: W      { public: int * y; void yf (int * yp) { y = yp; w = yp; }};
class Z: X, Y   { public: int * z; void zf (int * zp) { z = zp; x = zp; y = zp; }};
void h () { int hi;  W * pw;  X * px;  Y * py;  Z * pz;
            pz = new W;
            (*pz).w = & hi;
            pz -> xf ((*pz).X::w); }
```

9. Описать условия включения механизма виртуальности для метода класса. Привести пример записи виртуальной функции и обращения к ней.

10. Написать функцию `g()` с параметром — контейнером-вектором указателей на элементы вещественного типа. Функция проматривает контейнер в прямом порядке и обнуляет текущее указываемое значение, если следущее значение отрицательно. Например, $[1, -2, 4, -6, -4, 5] \to [0, -2, 0, 0, -4, 5]$. Далее функция в обратном порядке печатает значения, на которые указывают элементы контейнера. Функция возвращает число изменённых значений.